PROG-3

SUBTOPIC 8

# FRAMEWORKS

*David Rizo Valero*

*English version by Juan Antonio Pérez*

Version 20131125


Department of Computer Languages and Systems

Universitat d'Alacant
Universidad de Alicante

# Contents

# Frameworks and libraries

- Libraries and frameworks add features not originally provided by the programming language.
- **Framework** (visit Wikipedia for more information):
  - Software providing generic functionality that can be selectively changed by user code (by implementing interfaces, deriving new classes from abstract ones, configuration files, etc.), thus providing application specific software.
  - A framework is something that calls your code.
- **Library**: collection of classes that perform more-or-less specific operations.
  - A library is something you call from your code.

# Frameworks

- They use the three basic mechanisms in OOP::
  - Encapsulation: implementation is hidden
  - Polymorphism: the same code is used for different object types
  - Inheritance: functionality is reused
    - The universe is divided into categories, object sets (classes), and subclasses. Subclasses reuse the implementation or interface of their base classes.
- This allows us to

**Program for an interface instead of to program for an implementation**

# Frameworks. Control inversion

- **Control inversion**
  - The framework calls our methods when it decides to do so.
  - Control is in the framework.
  - Framework's users indicate the methods which have to be called, usually by giving an implementation for all the methods declared in interfaces or abstract classes.
  - The **Hollywood principle** (a software design methodology that takes its name from the cliché response given to amateurs auditioning in Hollywood) is followed: "Don't call us, we'll call you".

# Frameworks. Control inversion example (*)

- Ask the user to enter new data:
  - From the command line:

```
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);
```

(*) inspired on http://martinfowler.com/bliki/InversionOfControl.html

# Frameworks. Control inversion example (*)

- Ask the user to enter new data:
  - From a dialog, using an hypothetical window management framework:

```
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();
```

(*) inspired on http://martinfowler.com/bliki/InversionOfControl.html

# Frameworks. Control inversion

- In the command line example, our code has total control over execution flow.
- This does not hold for the dialog case. Part of the execution flow is controlled by the window framework.
- Control has been inverted. The framework is calling our methods.

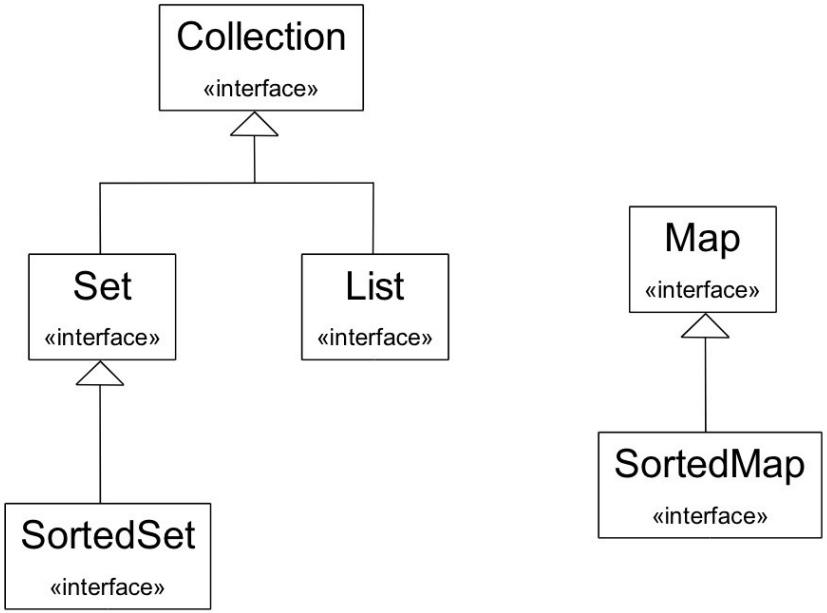
# Frameworks. Control inversion

- Example: **JUnit** is a framework.
  - It works by control inversion. Users define a set of methods to call (methods setUp(), testSomething(),...) and the framework decides when to call them:
    - Start testing process
    - Load classes containing tests
    - Invoke methods @BeforeClass
    - For each test:
      - Invoke @Before methods before each test
      - Invoke test method
      - Invoke @After methods after each test
    - etc...
- JUnit is controlling the execution flow.


# Java Collection Framework (JCF)

# Java Collection Framework (JCF)

- Set of classes and interfaces included in JDK that implement commonly reusable collection data structures: stacks, queues, vectors, maps, etc.
- The JCF provides both interfaces and classes that implement them, but programmers can also write their own implementation.
- Although it is a framework, it also works in a manner of a library.
- Since JDK 1.5, .



# JCF

```
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.
}
```

# JCF

```
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();
}
```

- Iterators

```
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

```
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();
}
```

# JCF

- The JCF provides both interfaces that define various collections and classes that implement them.
  - If only those implementations are used, the JFC will be used as a library and not as a framework.
  - HashSet is much faster than TreeSet (constant-time versus log-time for most operations like add, remove and contains) but offers no ordering guarantees like TreeSet.

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

# JCF: algorithms

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

- JCF does not include a reference implementation of Comparator

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

# JCF: algorithms

- Use of the framework as a library:

  (this code prints: [2,7,9])

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

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

- Use as a framework (interface implementation):

```
ArrayList<MiClase> v = ..... // initialization
// anonymous implementation of interface Comparator
Comparator c = new Comparator<MiClase>() {
  @Override
  public int compare(MiClase arg0, MiClase arg1) {
        /* code for comparing objects */
  }
};
// JFC calls our compare method to sort the elements of
// the list
Collections.sort(v);
```

# Framework example

# Framework example

**Car racing simulation**

- Toy framework
  - Not necessarily for cars
- Framework includes out of the box:
  - Circuit definition
  - Simulation of the race process
- How to use it
  - Indicate which cars will take part and the number of loops.
  - Extend and implement the following classes according to the (hypothetical) documentation:

```
Vehiculo, ICorredor, ICocheAuxiliar
```

# Framework example

**Interfaces**

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

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

# Framework example

**Class Vehiculo**

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

        private String marca;
}
```

# Framework example

**Class Circuito**

```
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>();
	}
...
```

# Framework example

**Class Circuito**

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

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

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

# Framework example

## Class 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!");
}}
```

# Framework example

**Using the framework**

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

# Framework example

**Using the 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;
}
```

# Framework example

**Using the 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;
}
```

# Framework example

**Using the framework**

```
class CamionFormula1 extends Formula1 {

        public CamionFormula1 (String marca) {
                super (marca);
        }

        public void dar_vuelta () {
          nvueltas++;
          System.out.println(
            "CamionFormula1[" +get_marca()+
            "], vuelta " +nvueltas);
        }
}
```

# Framework example

**Using the framework**
**Main program**

```
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");
...
```

# Framework example

**Using the framework**
**Main program**

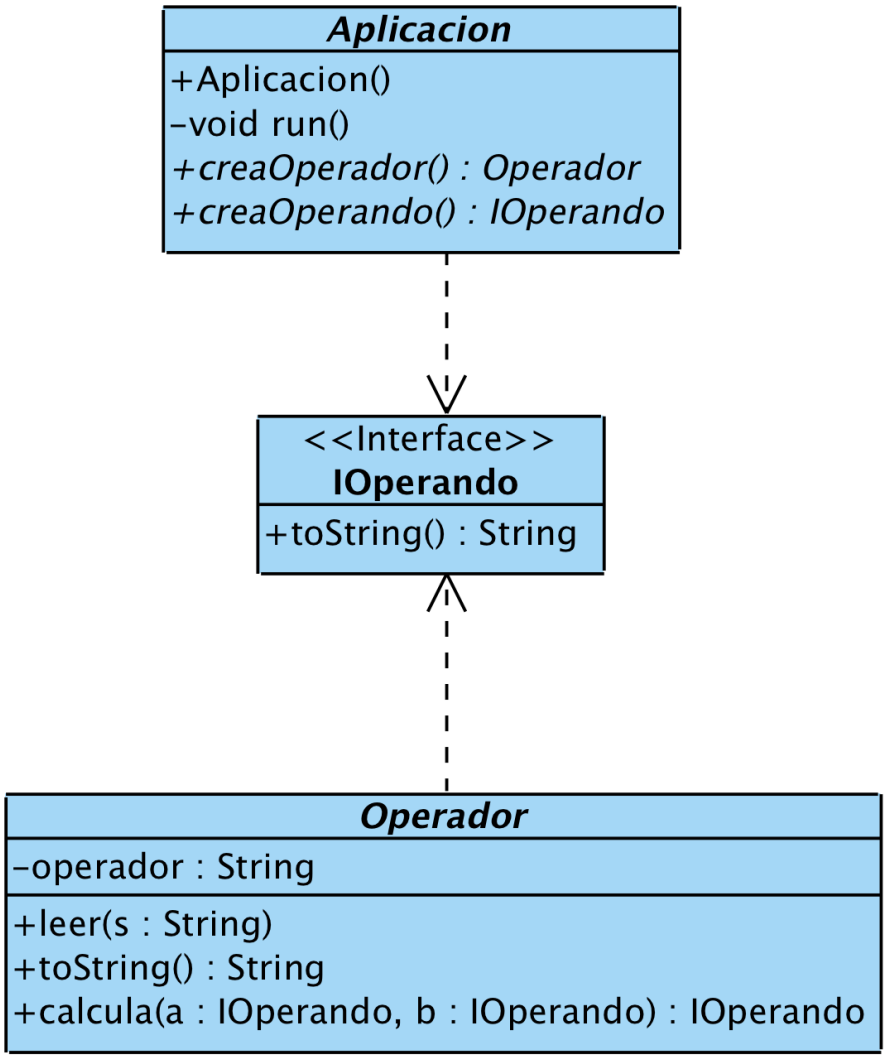
```
...
  for (int n = 0; n < MAXCOCHES; n++) {
    int rn = (new Random()).nextInt(100);
    if (rn < 50) { // Formula1
      String marca = "HRT"+n;
      c.add_vehiculo (new Formula1(marca));
    } else {
      String marca = "RENAULT"+n;
      c.add_vehiculo (new CamionFormula1 (marca));
    }
 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

# Another framework example

- Let's develop our own *toy* framework to perform binary operations.
  - Operands need not to be necessarily numbers.
- Our framework will provide:
  - Operand reading
  - Operator reading
  - Result output
  - Different verifications
- How to use it
  - By extending classes and implementing interfaces following the guidelines in the documentation (not shown here):

```
Aplicacion, IOperando,Operador
```

# Framework example



# Framework example

```
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();
..........
```

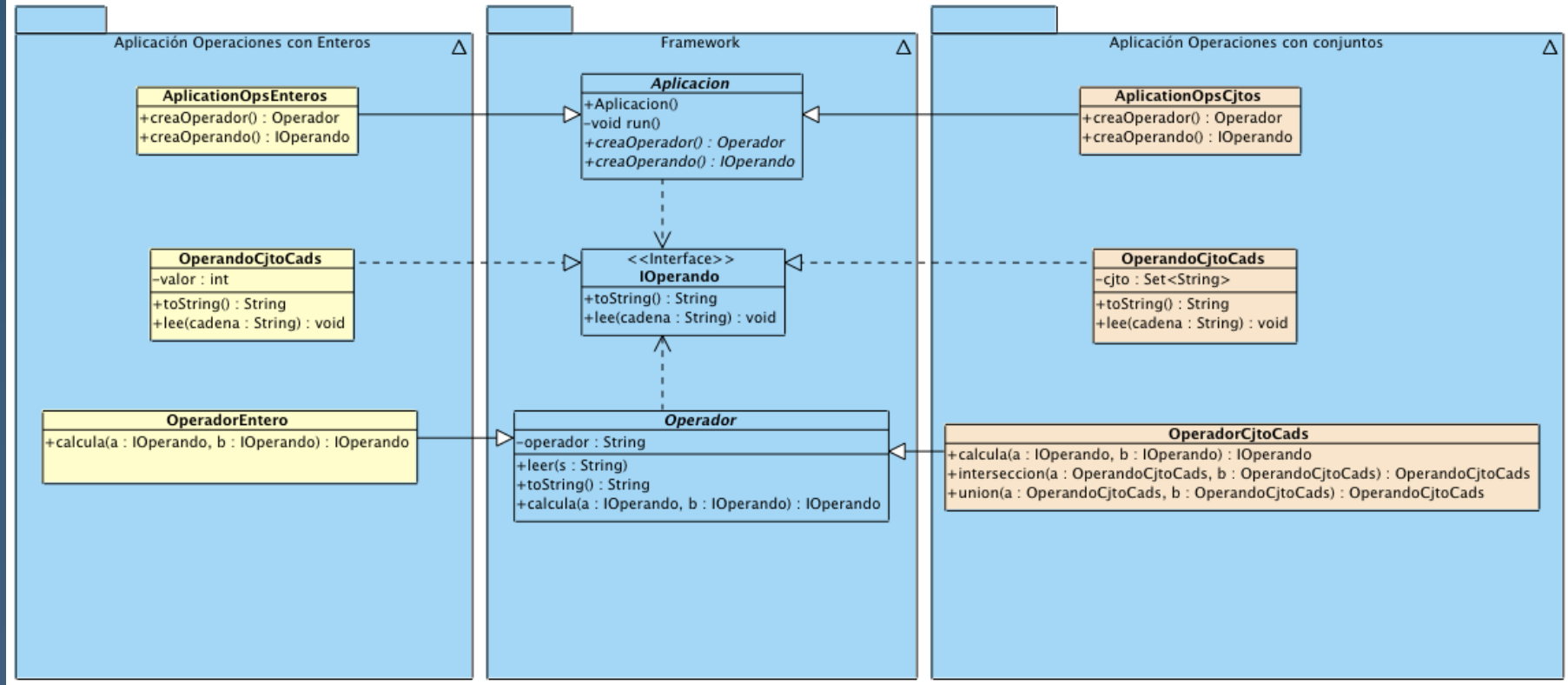
# Framework example

- The framework will call methods in user's classes (those inheriting from Aplicacion, Operador and implementing 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 (ExcepcionFramworkOps e) {
        System.err.println("Error: " + e.getMessage());
 }
}
```

# Framework example

- Two applications built with this framework: operations with integers and operations with sets



# Framework example

- Building an application to operate with integers:

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

@Override
public OperandoEntero creaOperando() {
 return new OperandoEntero();
}
}
```

# Framework example

- Extending Operador

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

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

}
```

# Framework example

- Implementing interface IOperando

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

# Framework example

- Example of execution:

```
Introduce el primer operando:
10
Introduce la operación:
-
Introduce el segundo operador:
4
El resultado de la operación
10-4=6
```

# Framework example

- Building an application to operate with string sets:

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

# Framework example

- Extending Operador

```
public class OperadorCjtoCads extends Operador<OperandoCjtoCads> {
 @Override
 public OperandoCjtoCads calcula(OperandoCjtoCads a, OperandoCjtoCads b)
         throws ExcepcionFramworkOps {
   if (toString().equals("U")) {
         return union(a,b);
   } else if (toString().equals("^")) {
         return interseccion(a,b);
   } else {
     throw new ExcepcionFramworkOps("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);
 }
}
```

# Framework example

- Implementing interface 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>();
 }
```

# Framework example

- Implementing interface IOperando (2/2)

```
/**
 * Reads the string with comma-separated elements
 */
@Override
public void lee(String cadena) throws ExcepcionFramworkOps {
 try {
        String [] cads = cadena.split(",");
        for (String string : cads) {
                cjto.add(string.trim());
        }
 } catch (NumberFormatException e) {
        throw new ExcepcionFramworkOps("Error leyendo el operador: "
         + e.getMessage());
 }
}
public String toString() {
        return cjto.toString();
}

}
```

# Framework example

- Example of execution:

```
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]U[b, d]=[a, b, c, d]
```

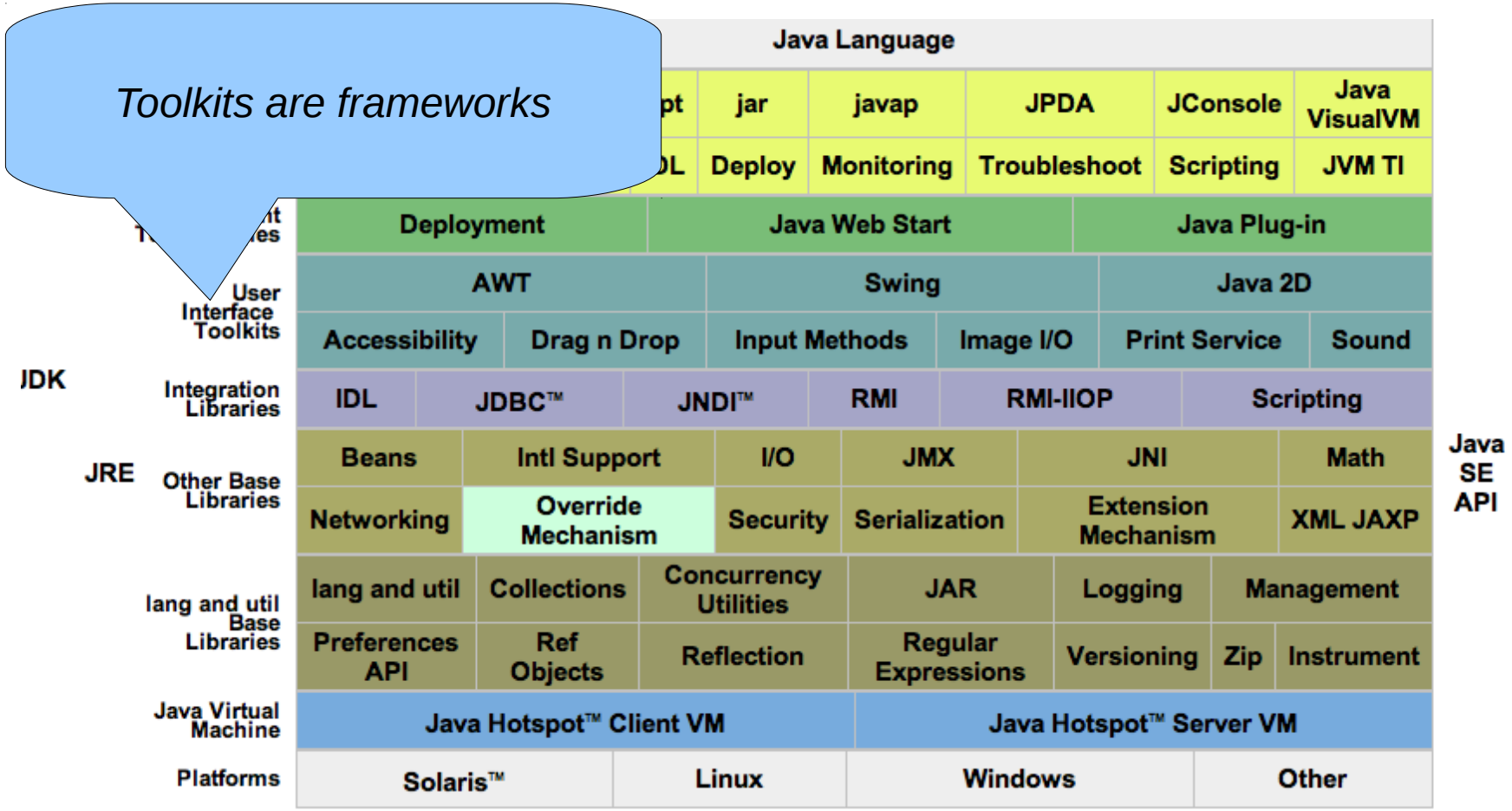
PROG-3

# JDK libraries and toolkits

# Libraries

- Collection of classes which may be used by our applications.
- They do no not require programmers to extend classes or implement interfaces.
- They can be seen as black boxes.
- The developer of the library provides an API (application program interface)
  - Collection of classes and their methods
- The reference implementation of the JCF included in JDK may be considered as a library:
  - java.util.ArrayList
  - java.util.Stack
  - java.util.TreeSet
  - ...
- In C++ similar libraries exist: STL

# JDK


Toolkits are frameworks
Java Language
jar
javap
JPDA
JConsole
Java VisualVM
Deploy
Monitoring
Troubleshoot
Scripting
JVM TI
Deployment
Java Web Start
Java Plug-in
User Interface Toolkits
AWT
Swing
Java 2D
Accessibility
Drag n Drop
Input Methods
Image I/O
Print Service
Sound
JDK
Integration Libraries
IDL
JDBC™
JNDI™
RMI
RMI-IIOP
Scripting
JRE
Other Base Libraries
Beans
Intl Support
I/O
JMX
JNI
Math
Networking
Override Mechanism
Security
Serialization
Extension Mechanism
XML JAXP
Java SE API
lang and util Base Libraries
lang and util
Collections
Concurrency Utilities
JAR
Logging
Management
Preferences API
Ref Objects
Reflection
Regular Expressions
Versioning
Zip
Instrument
Java Virtual Machine
Java Hotspot™ Client VM
Java Hotspot™ Server VM
Platforms
Solaris™
Linux
Windows
Other

PROG-3

# JDBC

# JDBC

- Java DataBase Connectivity
- Actually, it is a framework used by RDBMS vendors to make easier the access to their systems from Java programs.
- JDK provides a collection of interfaces and abstract classes which vendors have to specialize.
- Developers directly use vendor libraries, which are known as *drivers* or connectors. Examples:
  - MySQL: mysql-connector-java-5.1.18-bin
  - Oracle: ojdbc6.jar
  - SQLServer: sqljdbc4.jar
  - ...
- The complete list of drivers can be found at http://developers.sun.com/product/jdbc/drivers

# JDBC

- Basic procedure:
  - 1st, create a JDBC connection
  - 2nd use the connection to make CRUD (create, read, update, delete) operations against the database
    - By using class Statement
  - 3º close the connection

```
Class.forName("com.mysql.jdbc.Driver"); // load driver
// Connection string
String dbURL = "jdbc:mysql://localhost/mibbdd";
// Connection
Connection con = DriverManager.getConnection(dbURL, "milogin", "mipassword");
// CRUD operations
con.close(); // close connection to database
```

# JDBC

- Insert / update

```
Statement stmt = conexion.createStatement(); // using java.sql.Statement instead of 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");
```

- Delete

```
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");
```

- Retrieve

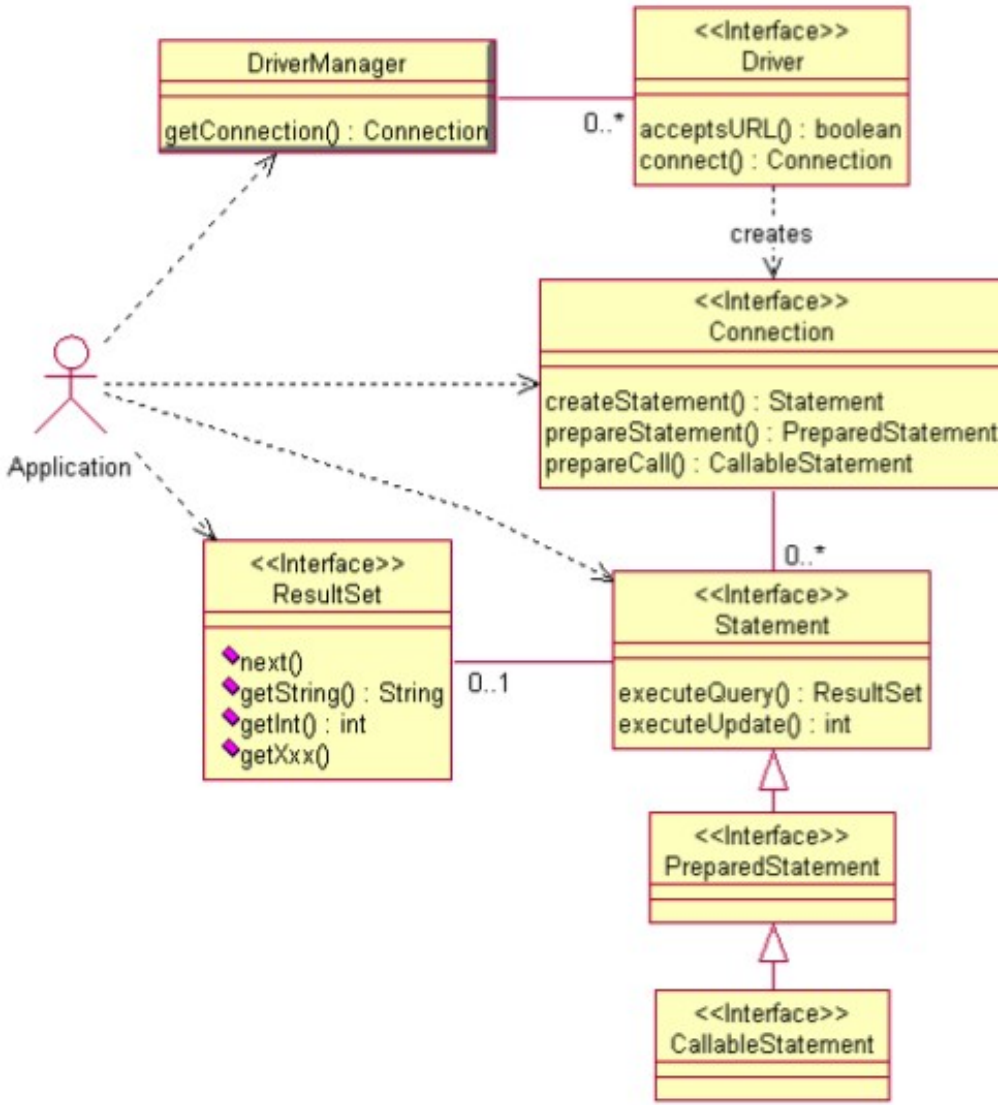
```
String sqlConsulta = "SELECT abreviatura, nombre from equipo order by abreviatura";
ResultSet rstEquipos = stmt.executeQuery(sqlConsulta);
while( rstEquipos.next() ) {
   String abrv = rstEquipos.getString("abreviatura");
   String nombreCompleto = rstEquipos.getString("nombre");
   System.out.println(abrv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

# JDBC

- JDBC includes mechanisms to improve performance
- For example, PreparedStatement
- SQL sentence is precompiled by the DBMS
- It is created upon a connection
- Sentences may contain variables marked with '?'

```
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);

}
```

# JDBC


DriverManager
getConnection() : Connection
<<Interface>>
Driver
acceptsURL() : boolean
connect() : Connection
0..*
creates
<<Interface>>
Connection
createStatement() : Statement
prepareStatement() : PreparedStatement
prepareCall() : CallableStatement
Application
<<Interface>>
ResultSet
next()
getString() : String
getInt() : int
getXxx()
0..1
0..*
<<Interface>>
Statement
executeQuery() : ResultSet
executeUpdate() : int
<<Interface>>
PreparedStatement
<<Interface>>
CallableStatement


# Processing XML

# XML processing with Java

- JDK allows for two different approaches to parse and process XML content:
  - DOM Parser: the whole XML document is read into an element tree in memory: it works as a library
  - SAX Parser: useful for very large documents
    - It works as a framework. Every time an opening or closing tag is read from the file, a **callback** method (provided by the user) is invoked

```
<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 are elements`
  - `width, height, cx, cy, fill` are attributes

# DOM Parser

```
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);

            //Methods defined in Document to process the DOM tree are used
here

   } catch (ParserConfigurationException e){
            //HANDLE EXCEPTION
   } catch (SAXException e){
            //HANDLE EXCEPTION
   } catch (IOException e){
            //HANDLE EXCEPTION
   }
```

# SAX Parser

- MiElementHandler is our own class inheriting from DefaultHandler

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

- Every time an opening tag is read, the method

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

defined in MiElementHandler will be invoked

PROG-3

# Logging

# Logging

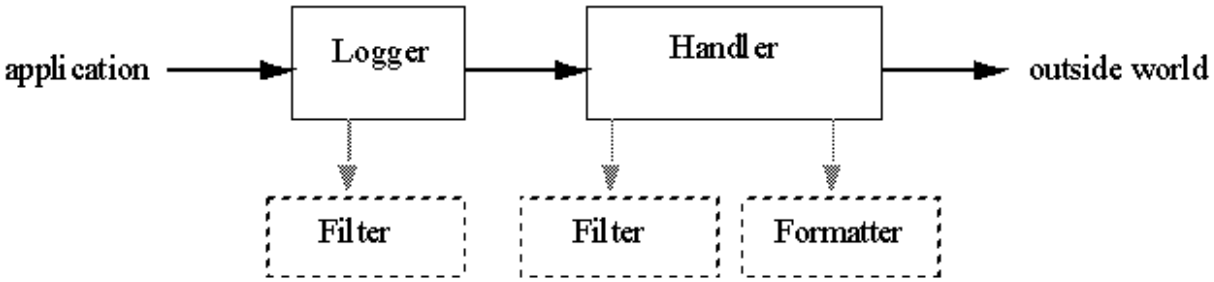
- Java logging (recording of activity) framework: it may be configured by means of code or files:
  - Inefficient and non-parameterizable option:

```
.....
System.out.println("paso por aquí y x=" + x);
.....
```

    All logs are mixed in the console...
  - More efficient and parameterizable option:

```
logger.info("paso por aquí y x=" + x);
```

- JDK includes the Logging framework:



# Logging

- Usage:

```
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");
```

- Console output:

```
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
```

# Logging

- To only show the logs from a particular level, the first lines of your program must include:

```
Logger.getLogger("").getHandlers()[0].setLevel(Level.SEVERE);

System.setProperty("java.util.logging.ConsoleHandler.level", "Level.OFF");
```

- To make it better, use a configuration file (*properties* file, using Java terminology)

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

  - -D passes properties values to the virtual machine
- It can also be done like this:

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

# Logging

```
# global log level
.level = WARNING

# output handlers: a file handler and a console
# handler in this case
handlers = java.util.logging.FileHandler,
   java.util.logging.ConsoleHandler

# configuration for the file handler
# log level
java.util.logging.FileHandler.level = ALL
# output file
java.util.logging.FileHandler.pattern = ./log/prog3.log
# maximum size (bytes)
java.util.logging.FileHandler.limit = 10485760
# maximum number of log files
java.util.logging.FileHandler.count = 3
# class responsible for formatting the output
java.util.logging.FileHandler.formatter =
   java.util.logging.XMLFormatter
# append output (if false, previous logs will be erased)
java.util.logging.FileHandler.append = true
```

# Logging

- File prog3.log generated with the previous configuration:

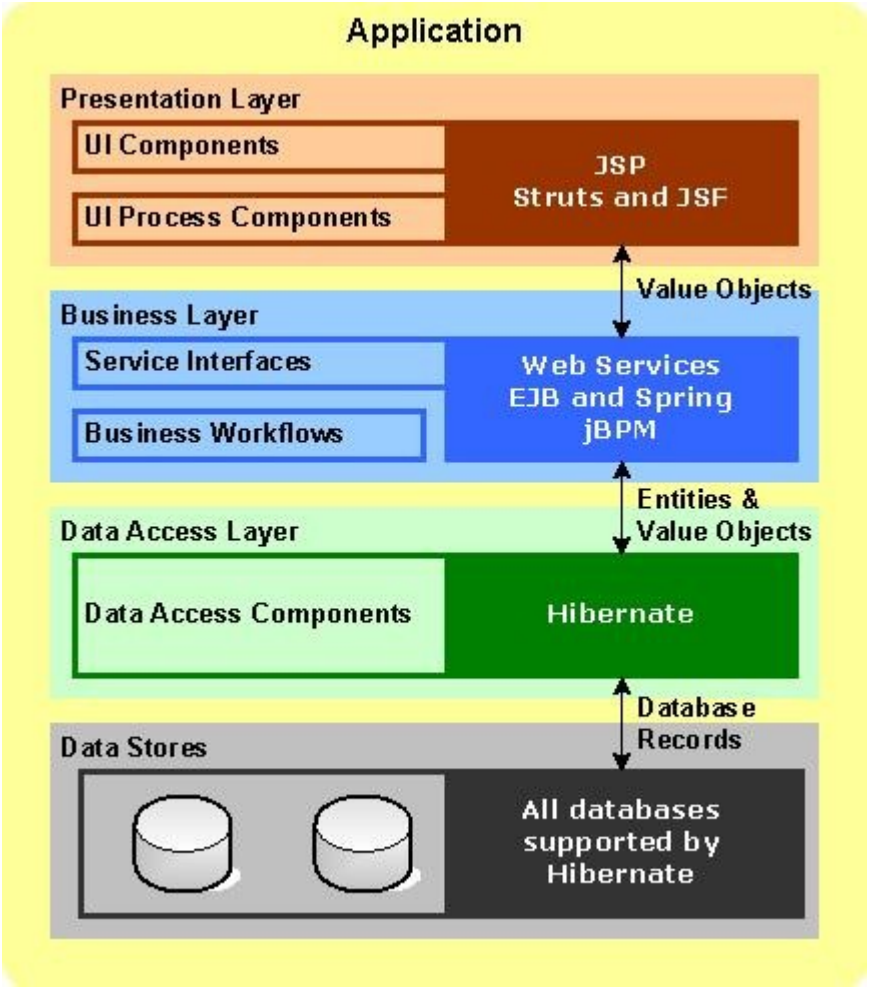
```
<?xml version="1.0" encoding="MacRoman" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2011-11-18T19:37:08</date>
  <millis>1321641428407</millis>
  <sequence>0</sequence>
  <logger>MiClase</logger>
  <level>WARNING</level>
  <class>MiClase</class>
  <method>F</method>
  <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>
```

PROG-3

# Hibernate

# Hibernate

- **Object-relational mapping (ORM) / persistence**
- Hibernate reads a set of configuration files (or Java annotations) and automatically generates classes for persistently storing, reading, etc. (CRUD operations) objects into a database.
- With JDBC, programmer needs to do this manually:
  - (insert into ...., delete from, .... update, ...., select ...)
  - Hibernate frees us from this responsibility



# Hibernate

- Configuration files will contain data such a as the name of the tables in the database or how our own data types correspond to database types:

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

- Making an object persistent is as simple as:

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

  - This will eventually execute a SQL sentence insert into...

PROG-3

# Apache Commons

# Apache Commons

- Collection of libraries from the Apache project http://commons.apache.org/
- The most useful ones are:
  - CLI: provides an API for parsing command line options passed to programs
  - Collections: adds extra data types
  - Configuration: reading configuration data from a variety of sources
  - Email
  - FileUpload: add robust, high-performance, file upload capability
  - Math:  a library of lightweight, self-contained mathematics and statistics components addressing things not available in the Java language
- E.g., MultiMap (map that holds a collection of values against each key) or MultiKeyMap (multiple keys to map to a value):

```
MultiKeyMap mapa=new ....;
MultiKey key = new MultiKey(partido.getLocal(), partido.getVisitante());
// adding:
this.mapa.put(key, partido);
.......
// retrieving
MultiKey key = new MultiKey(local, visitante);
Partido p = (Partido) mapa.get(key);
```

PROG-3

# Other Java frameworks

# Other Java frameworks

- GWT (Google Web Toolkit): it compiles Java code into Javascript to easily build web applications
- Spring: application development of enterprise applications
  - HTML5, REST, AJAX, mobile devices
  - object-relational mapping
  - Integration with social platforms
  - Security...
- Apache Struts: framework for web application development
- JUnit: unit testing
- JavaFX: it was a framework for creating and delivering web applications; Oracle has turn it into something completely different
- Apache Lucene: high-performance, full-featured text search engine library
- **Libraries and frameworks discussed in these slides are all open-source!**