PROG3-UD9 - Frameworks

Figura 1: C++ Input/Output Stream Library

(véase http://www.cplusplus.com/reference/)

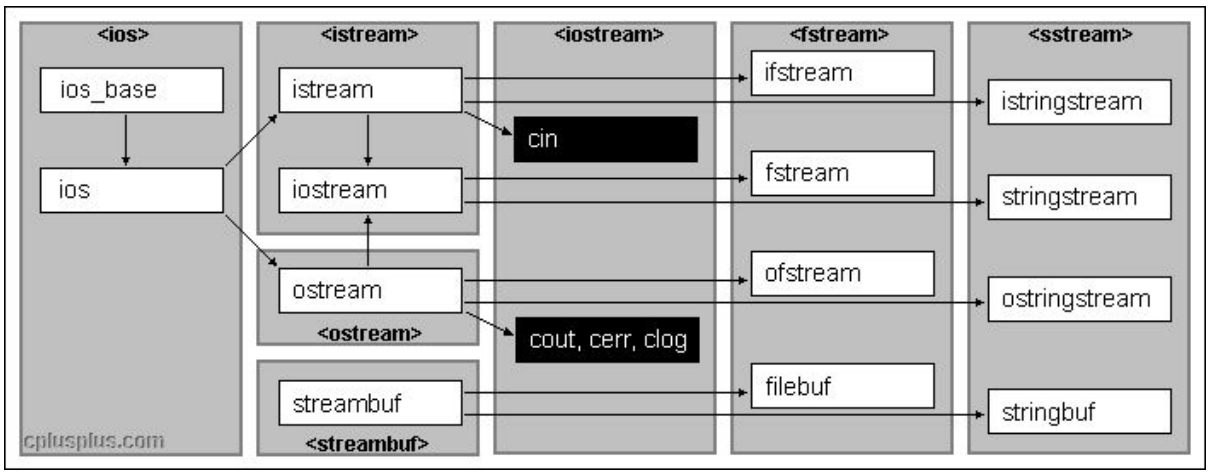


Figura 2: Java Collection Framework (JCF)

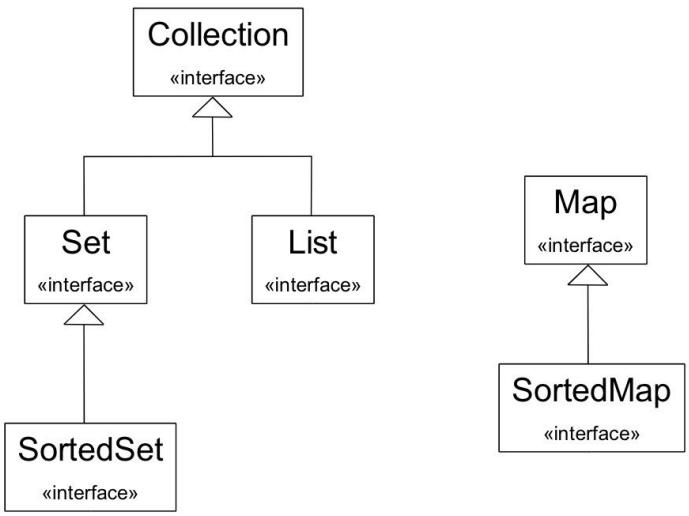


Figura 3. Interfaces de la JCF

```
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.
}
```

Figura 4. Algoritmos en la JCF

```
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...
}
```

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

Figura 5. Uso de la comparación

a- Uso del JCF como librería

```
ArrayList<Integer> v = new ArrayList<>();
v.add(10);
v.sort();
```

b- Uso del JCF como framework

```
class ComparadorCoordenada implements Comparator<Coordenada>
{
      public int compareTo(Coordenada otro) {
            int result = x - otro.x;
            if (result == 0) {
                  result = y - otro.y;
            }
            return result;
      }
}
ArrayList<Coordenada> v = new ArrayList<>();
…..
Collections.sort(v, new ComparadorCoordenada()) {

}
```

Figura 6. Uso de clases anónimas

```
ArrayList<Coordenada> v = new ArrayList<>();
…..
Collections.sort(v, new Comparator<Coordenada> {
      public int compareTo(Coordenada otro) {
            int result = x - otro.x;
            if (result == 0) {
                  result = y - otro.y;
            }
            return result;
      }
});
```

Figura 7. CRUD (create, retrieve, update, delete) en JDBC

Estructura básica

1. Crear conexión
2. Usar la conexión para consultar / manipular la BBDD
3. cerrar la conexión

```
Class.forName("com.mysql.jdbc.Driver"); // cargamos driver
// Cadena de conexiónn
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

```
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

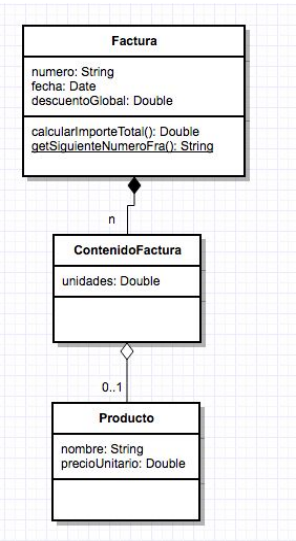
```
Statement stmt = conexion.createStatement();
String sqlBorrado = "delete from equipo where abreviatura = 'ALC' or abreviatura = 'HER'";
int filasBorradas = stmt.executeUpdate(sqlBorrado);
System.out.println("Se han borrado " + filasBorradas + " registros");
```

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);
}
```

Figura 8. Mapeo Objeto Relacional

UML



E/R

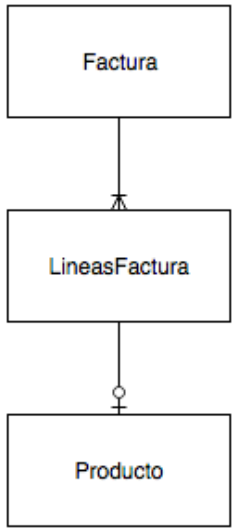


Figura 9. Uso del framework de mapeo O/R Hibernate

Fichero hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <mapping class="es.ua.dlsi.prog3.facturas.Factura"/>
    <mapping class="es.ua.dlsi.prog3.facturas.Cliente"/>
    <mapping class="es.ua.dlsi.prog3.facturas.ContenidoFactura"/>
  </session-factory>
</hibernate-configuration>
```

Clases de Java

Factura.java

```
@Entity
class Factura {
        @Id
        private Long id;
        private Set<ContenidoFactura>
contenidos;

        @OneToMany (mappedBy = "factura")
        public Set<ContenidoFactura>
getContenidos() {
            return this.contenidos;
        }
        /* Resto de métodos ... */
}
```

Producto.java

```
@Entity
class Producto {
        @Id
        private Long id;
        private String nombre;
        private Double precioUnitario;
}
```

```
        /* Resto de métodos, getters,
setters....*/
}
```

ContenidoFactura.java

```
@Entity @Table(name = "LineasFactura")
public class ContenidoFactura  {
        @Id
        private Long id;
        private Factura factura;
        private Producto producto;

        @ManyToOne
        public Factura getFactura() {
                return this.factura;
        }
        @ManyToOne
        public Producto getProducto() {
                return this.producto;
        }
        /* Resto de métodos, getters,
setters....*/
}
```

Inserción de una factura (esqueleto)

```
        ...
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Factura factura = new Factura();
        factura.setNumero(Factura.getSiguienteNumeroFactura());
        session.save(temporada);
        factura.add(new ContenidoFactura(productoEnOferta));
        session.save(factura);
        session.getTransaction().commit();
```

…

Figura 10. Framework “Simulador de carreras”

- Framework propio (de juguete) para la realización de carreras (de coches,...)
- El framework nos da hecho:
    - La definición de un circuito
    - La simulación del proceso de la carrera
- Cómo usarlo
    - Especificar qué vehículos van a intervenir y cuántas vueltas hay que dar.
    - Extendiendo e implementando las clases que se nos indica en la documentación (hipotética)
- Clases / interfaces: Vehiculo, ICorredor, ICocheAuxiliar

Definición del framework

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

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

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

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>();
    }
       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; }
```

```
        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);
    }
}

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;
}
```

```
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;
}

class CamionFormula1 extends Formula1 {

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

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

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");
       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
```

Figura 10. Inyección de código con el framework GUICE

```
interface ICalculadoraOfertas {
	List<Oferta> calculaOfertas(List<Articulo> articulosEnTicket);
}

class Oferta3x2 implements ICalculadoraOfertas {…}

class OfertaDcto70SegundaUnidad implements ICalculadoraOfertas {…}
class Ticket {
	List<Articulo> articulos;
	public double calculaImporteTotal(ICalculadoraOfertas calc) {	…
		List<Oferta> ofertas = calc.calculaOfertas(articulos);	…
	}
}
class Principal {
	void metodoPrincipal() {
		Ticket ticket;
		…. inserción de artículos…
		[ aquí tenemos varias opciones para invocar al cálculo del importe total ]
	}
}
```

Opciones para invocar al cálculo del importe total:

1. Directamente con new
   `ticket.calculaImporteTotal(new Oferta3x2())`
2. Usando una factory
   ```
   class FactoryCalcOfertas {
   	…
   	public static ICalculadoraOfertas createCalcOfertas() {
   	Creación de ICalculadoraOfertas - podemos cargar el nombre de la clase a crear desde un fichero e instanciarlo usando reflexión
   	}
   }
   ```
3. Usando la inyección de dependencias. Utilizamos el framework GUICE (https://github.com/google/guice)
   a. De manera muy simplificada, crearemos una clase de configuración que tenga este código
      `bind(ICalculadoraOfertas.class).to(Oferta3x2.class)`
      i. En lugar de escribir Oferta3x2.class, eso se puede cargar desde un fichero. P.ej.
         1. `String nombreClaseOferta = <valor cargado desde un fichero>`
         2. `Class<?> claseOfertas = Class.forName(nombreClaseOferta)`
         3. `bind(ICalculadoraOfertas.class).to(claseOfertas)`

   b. De esta forma, siempre que esté declarado `ICalculadoraOfertas` , el inyector de dependencias GUICE reemplazará `ICalculadoraOfertas` por `Oferta3x2`, nosotros no tendremos que instanciar nada (principio Hollywood, “no nos llames, nosotros te llamaremos a tí”