USING JAVA REFLECTION

- La reflexión permite que un programe se examine a sí mismo y manipule sus propiedades internas.

- Un uso tangible de la reflexión es en JavaBeans, donde los componentes de software se pueden manipular visualmente a través de una herramienta de construcción.

Ejemplo:

```
import java.lang.reflect.*;

   public class DumpMethods {
      public static void main(String args[])
      {
         try {
            Class c = Class.forName(args[0]);
            Method m[] = c.getDeclaredMethods();
            for (int i = 0; i < m.length; i++)
            System.out.println(m[i].toString());
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

Configuración para usar reflexión

- Las clases de reflexión, se encuentran en java.lang.reflect. Hay que seguir tres pasos para usarlas:

    - Obtener un objeto java.lang.Class (representa clases e interfaces en un programa Java en ejecución) para la clase que se quiere manipular. Esto se puede hacer de la siguiente manera:

        - `Class c = Class.forName("java.lang.String");` para obtener el objeto para string.

        - `Class c = int.class;`

        - `Class c = Integer.TYPE;`

- Llamar a un método como getDeclaredMethods para obtener una lista con todos los métodos declarados por la clase.

- Usar la API de reflexión para manipular la información.

Simulando el operador instanceOf

- Class.isInstance se puede usar para simular el operador instanceOf:

Ejemplo:

```
class A {}

   public class instance1 {
      public static void main(String args[])
      {
         try {
            Class cls = Class.forName("A");
            boolean b1
              = cls.isInstance(new Integer(37));
            System.out.println(b1);
            boolean b2 = cls.isInstance(new A());
            System.out.println(b2);
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

Descubrir métodos de una clase

- Uno de los usos más basicos de reflexión es averiguar qué métodos se definen dentro de una clase. Para eso se puede utilizar el siguiente código:

Ejemplo:

```
import java.lang.reflect.*;

   public class method1 {
      private int f1(
       Object p, int x) throws NullPointerException
      {
         if (p == null)
            throw new NullPointerException();
         return x;
      }

      public static void main(String args[])
      {
         try {
           Class cls = Class.forName("method1");
```

```
            Method methlist[]
              = cls.getDeclaredMethods();
            for (int i = 0; i < methlist.length;
               i++) {
               Method m = methlist[i];
               System.out.println("name
                 = " + m.getName());
               System.out.println("decl class = " +
                                  m.getDeclaringClass());
               Class pvec[] = m.getParameterTypes();
               for (int j = 0; j < pvec.length; j++)
                  System.out.println("
                   param #" + j + " " + pvec[j]);
               Class evec[] = m.getExceptionTypes();
               for (int j = 0; j < evec.length; j++)
                  System.out.println("exc #" + j
                    + " " + evec[j]);
               System.out.println("return type = " +
                                  m.getReturnType());
               System.out.println("-----");
            }
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

- Si se usa getMethods en lugar de getDeclaredMethods se puede también obtener información de métodos heredados.

<u>Obteniendo información sobre constructores</u>

- Una forma similar se utiliza para buscar los constructores de una clase:

<u>Ejemplo</u>:

```
import java.lang.reflect.*;

   public class constructor1 {
      public constructor1()
      {
      }

      protected constructor1(int i, double d)
      {
      }

      public static void main(String args[])
      {
         try {
           Class cls = Class.forName("constructor1");
```

```
          Constructor ctorlist[]
              = cls.getDeclaredConstructors();
          for (int i = 0; i < ctorlist.length; i++) {
              Constructor ct = ctorlist[i];
              System.out.println("name
                = " + ct.getName());
              System.out.println("decl class = " +
                          ct.getDeclaringClass());
              Class pvec[] = ct.getParameterTypes();
              for (int j = 0; j < pvec.length; j++)
                 System.out.println("param #"
                    + j + " " + pvec[j]);
              Class evec[] = ct.getExceptionTypes();
              for (int j = 0; j < evec.length; j++)
                 System.out.println(
                   "exc #" + j + " " + evec[j]);
              System.out.println("-----");
            }
           }
           catch (Throwable e) {
              System.err.println(e);
           }
      }
   }
```

Buscando información sobre campos de clase

- Para saber qué campos son definidos en la clase, se puede usar el siguiente código:

Ejemplo:

```
import java.lang.reflect.*;

   public class field1 {
      private double d;
      public static final int i = 37;
      String s = "testing";

      public static void main(String args[])
      {
         try {
            Class cls = Class.forName("field1");

            Field fieldlist[]
              = cls.getDeclaredFields();
            for (int i
              = 0; i < fieldlist.length; i++) {
               Field fld = fieldlist[i];
               System.out.println("name
                  = " + fld.getName());
               System.out.println("decl class = " +
                           fld.getDeclaringClass());
               System.out.println("type
                  = " + fld.getType());
               int mod = fld.getModifiers();
```

```
            System.out.println("modifiers = " +
                       Modifier.toString(mod));
            System.out.println("-----");
          }
        }
        catch (Throwable e) {
           System.err.println(e);
        }
     }
}
```

- Modifier es una clase de reflexión que representa los modificadores encontrados en un miembro de campo, como 'private int'.
- Los modificadores son representados por un entero. Modifier.toString es utilizado para devolver una representación de cadena en el orden oficial de declaración "oficial".
- getDeclaredFields obtiene información solo de los campos declarados en una clase. getFields también obtiene información sobre campos definidios en superclases.

Invocando métodos por nombre

- Se puede utilizar reflexión para invocar métodos de un nombre específico.

Ejemplo:

```
import java.lang.reflect.*;

   public class method2 {
      public int add(int a, int b)
      {
         return a + b;
      }

      public static void main(String args[])
      {
         try {
           Class cls = Class.forName("method2");
           Class partypes[] = new Class[2];
            partypes[0] = Integer.TYPE;
            partypes[1] = Integer.TYPE;
            Method meth = cls.getMethod(
              "add", partypes);
            method2 methobj = new method2();
            Object arglist[] = new Object[2];
            arglist[0] = new Integer(37);
            arglist[1] = new Integer(47);
            Object retobj
              = meth.invoke(methobj, arglist);
            Integer retval = (Integer)retobj;
            System.out.println(retval.intValue());
         }
         catch (Throwable e) {
            System.err.println(e);
         }
```

```
      }
   }
```

Creando nuevos objetos

Ejemplo:

```
import java.lang.reflect.*;

   public class constructor2 {
      public constructor2()
      {
      }

      public constructor2(int a, int b)
      {
         System.out.println(
           "a = " + a + " b = " + b);
      }

      public static void main(String args[])
      {
         try {
           Class cls = Class.forName("constructor2");
           Class partypes[] = new Class[2];
            partypes[0] = Integer.TYPE;
            partypes[1] = Integer.TYPE;
            Constructor ct
              = cls.getConstructor(partypes);
            Object arglist[] = new Object[2];
            arglist[0] = new Integer(37);
            arglist[1] = new Integer(47);
            Object retobj = ct.newInstance(arglist);
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

- Esto encuentra un constructor que maneja los tipos de parámetros especificados y lo invoca.

Cambiando valores de campos

- El valor de esto es que deriva de la naturaleza dinámica de la reflexión, donde se puede buscar un campo por su nombre en un programa en ejecución y luego cambiar su valor.

Ejemplo:

```
import java.lang.reflect.*;

   public class field2 {
      public double d;

      public static void main(String args[])
```

```
      {
         try {
            Class cls = Class.forName("field2");
            Field fld = cls.getField("d");
            field2 f2obj = new field2();
            System.out.println("d = " + f2obj.d);
            fld.setDouble(f2obj, 12.34);
            System.out.println("d = " + f2obj.d);
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

Usando arreglos

- Los arreglos en Java son un tipo especializado de clase y una referencia del arreglo puede asignarse a una referencia de un objeto.

Ejemplo:

```
import java.lang.reflect.*;

   public class array1 {
      public static void main(String args[])
      {
         try {
            Class cls = Class.forName(
              "java.lang.String");
            Object arr = Array.newInstance(cls, 10);
            Array.set(arr, 5, "this is a test");
            String s = (String)Array.get(arr, 5);
            System.out.println(s);
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

Ejemplo más complejo:

```
import java.lang.reflect.*;

   public class array2 {
      public static void main(String args[])
      {
         int dims[] = new int[]{5, 10, 15};
         Object arr
           = Array.newInstance(Integer.TYPE, dims);

         Object arrobj = Array.get(arr, 3);
         Class cls =
           arrobj.getClass().getComponentType();
         System.out.println(cls);
         arrobj = Array.get(arrobj, 5);
         Array.setInt(arrobj, 10, 37);
```

```
        int arrcast[][][] = (int[][][])arr;
        System.out.println(arrcast[3][5][10]);
    }
}
```