Java Reflection es quizás el API que más versatilidad aporta al lenguaje Java ya que nos permite resolver muchos problemas de una forma totalmente diferente a la habitual. El API de Java reflection nos permite leer los metadatos de nuestras clases y trabajar con ellos. En un principio no es nada sencillo de entender .Vamos a construir un ejemplo que nos clarifique las ideas. Imaginemos que tenemos varias clases que identifican tipos de productos. En nuestro caso pueden ser Ordenador y Lavadora.

```
package com.arquitecturajava;

public class Ordenador {

private String id;
private String descripcion;
private int potencia;
public String getId() {
return id;
}
public void setId(String id) {
this.id = id;
}
public String getDescripcion() {
return descripcion;
}
public void setDescripcion(String descripcion) {
this.descripcion = descripcion;
}
public int getPotencia() {
return potencia;
}
```

```
public void setPotencia(int potencia) {
this.potencia = potencia;
}
public Ordenador(String id, String descripcion, int potencia) {
super();
this.id = id;
this.descripcion = descripcion;
this.potencia = potencia;
}

}
```

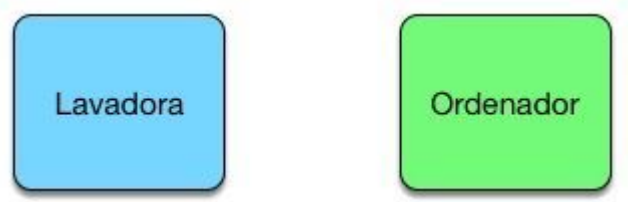
```
package com.arquitecturajava;

public class Lavadora {
private String id;
private String modelo;
private String descripcion;

public Lavadora(String id, String modelo, String descripcion) {
super();
this.id = id;
this.modelo = modelo;
this.descripcion = descripcion;
}
public String getId() {
return id;
}
```

```
public void setId(String id) {
this.id = id;
}
public String getModelo() {
return modelo;
}
public void setModelo(String modelo) {
this.modelo = modelo;
}
public String getDescripcion() {
return descripcion;
}
public void setDescripcion(String descripcion) {
this.descripcion = descripcion;
}
}
```

Ambas clases disponen de las propiedades descripción e id. Sin embargo las clases no están relacionadas de ninguna forma. No son hijas de la clase Producto ni implementan un interface Producto.



Así que trabajamos con ellas de forma independiente.

```
package com.arquitecturajava;
```

```
import java.util.ArrayList;
import java.util.List;

public class Principal {

        public static void main(String[] args) {
List&amp;amp;amp;amp;amp;lt;Ordenador&amp;amp;amp;amp;amp;gt;
lista = new
ArrayList&amp;amp;amp;amp;amp;lt;Ordenador&amp;amp;amp;amp;amp
;gt;();

                Ordenador o1 = new Ordenador("A1", "Ordenador gaming",
4);
                Ordenador o2 = new Ordenador("B1", "Ordenador
ofimatica", 2);
                lista.add(o1);
                lista.add(o2);

                for (Ordenador o : lista) {

                        System.out.println(o.getId());
                        System.out.println(o.getPotencia());
                        System.out.println(o.getDescripcion());

                }

                Lavadora l1= new Lavadora("L1","Standard","Lavadora
normal");
                Lavadora l2= new Lavadora("L2","VIP","Lavadora
avanzada");
```

```
List&amp;amp;amp;amp;amp;amp;lt;Lavadora&amp;amp;amp;amp;amp;amp;gt;
lista2= new
ArrayList&amp;amp;amp;amp;amp;amp;lt;Lavadora&amp;amp;amp;amp;amp;amp;
gt;();
                lista2.add(l1);
                lista2.add(l2);

                for (Lavadora l : lista2) {

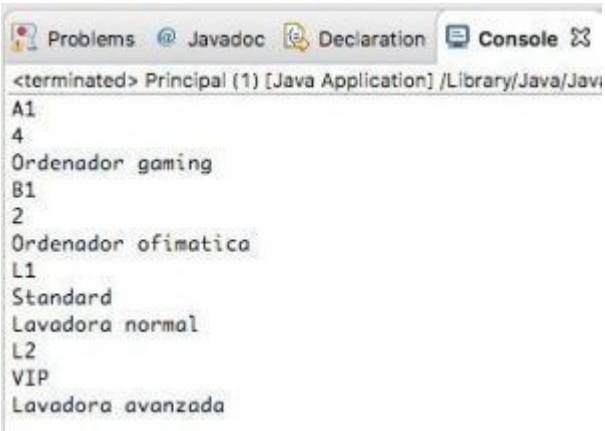
                        System.out.println(l.getId());
                        System.out.println(l.getModelo());
                        System.out.println(l.getDescripcion());

                }

        }

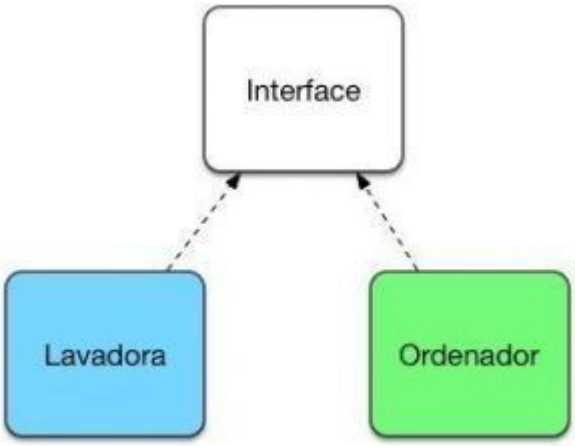
}
```

Creamos dos listas e imprimimos la información por la consola:

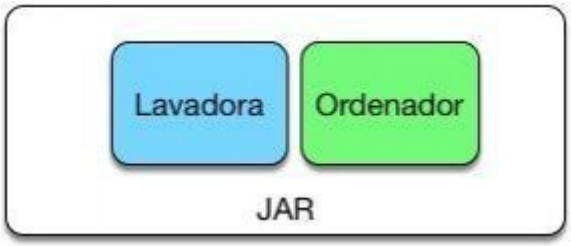


No nos queda más remedio que recorrerlas una a una lo cual es un problema. Las soluciones

más habituales a estos casos es crear interfaces o usar clases abstractas. De tal forma que a través del polimorfismo podamos trabajar de la misma forma con tipos muy dispares.



El problema comienza cuando por ejemplo tenemos cientos de tipos diferentes y NO disponemos del código fuente.



Esto es un verdadero problema ya que nos posible modificar las clases existentes. ¿Cómo podemos abordar este problema y tratar de las misma forma las Lavadores y los Ordenadores y otros tipos?

Java Reflection

Es aquí donde el API de reflection nos puede rescatar leyendo los metadatos de nuestros objetos e invocando los métodos necesarios .

```
package com.arquitecturajava;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.List;

public class Principal2 {

        public static void main(String[] args) {
List&amp;amp;amp;amp;amp;lt;Ordenador&amp;amp;amp;amp;amp;gt;
lista = new
ArrayList&amp;amp;amp;amp;amp;lt;Ordenador&amp;amp;amp;amp;amp
;gt;();

                Ordenador o1 = new Ordenador("A1", "Ordenador gaming",
4);
                Ordenador o2 = new Ordenador("B1", "Ordenador
ofimatica", 2);
                lista.add(o1);
                lista.add(o2);

                Lavadora l1= new Lavadora("L1","Standard","Lavadora
normal");
                Lavadora l2= new Lavadora("L2","VIP","Lavadora
avanzada");

List&amp;amp;amp;amp;amp;lt;Lavadora&amp;amp;amp;amp;amp;gt;
lista2= new
ArrayList&amp;amp;amp;amp;amp;lt;Lavadora&amp;amp;amp;amp;amp;
gt;();
```

```
                lista2.add(l1);
                lista2.add(l2);

                imprimirListaCualquiera(lista);
                imprimirListaCualquiera(lista2);

        }

        public static void
imprimirListaCualquiera(List&amp;amp;amp;amp;amp;lt;?&amp;amp;amp;
amp;amp;amp;gt; lista) {

                for(Object c: lista) {

                        Method[] metodos=c.getClass().getMethods();
                        for(Method m: metodos) {

                                //System.out.println(m.getName());

                                if (m.getName().equals("getId") ||
m.getName().equals("getDescripcion")) {

                                        try {
                                                String cadena=(String)
m.invoke(c, null);
System.out.println(cadena);
                                        } catch
(IllegalAccessException | IllegalArgumentException
                                                                |
InvocationTargetException e) {
                                                // TODO Auto-generated
```

```
catch block
                                                    e.printStackTrace();
                                                }
                                            }

                                    }

                                }

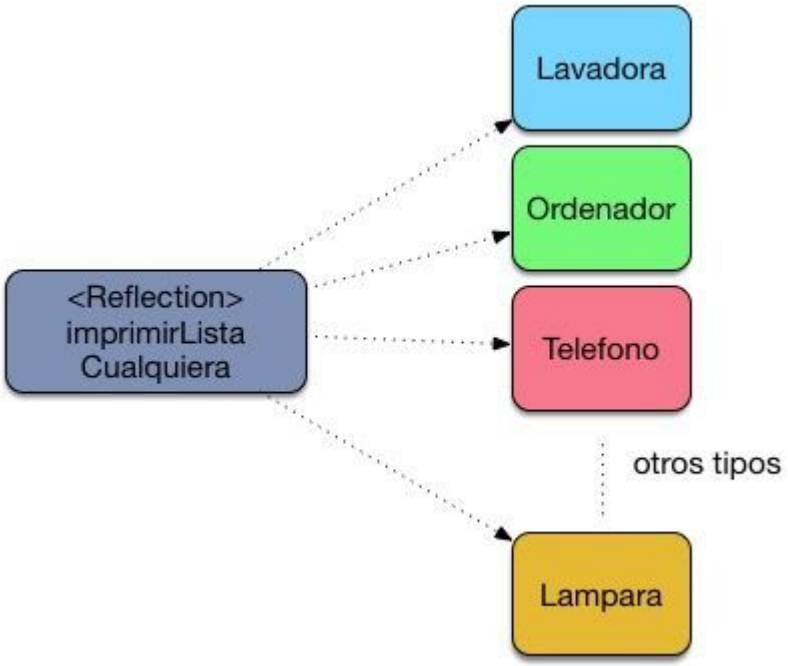
                        }
```

En este caso generamos un método “imprimirListaCualquiera” que accede al API de reflection y lee los metadatos de cada objeto.

1. En primer lugar utiliza un bucle para recorrer el Array y lee la lista de métodos que cada objeto contiene con reflection y getMethods();
2. En segundo lugar verifica que el objeto dispone de los métodos getId()y getDescripcion() usando m.getName().
3. Por último usa el método invoke del API de reflection para invocar el método que corresponda.
   El resultado por la consola será similar, eso sí en nuestro caso solo hemos solicidado getId() y getDescripcion():

```
Problems  Javadoc  Declaration
<terminated> Principal2 (1) [Java Application]
A1
Ordenador gaming
B1
Ordenador ofimatica
L1
Lavadora normal
L2
Lavadora avanzada
```

La ventaja es que con construir un único método ha sido suficiente .



Para poder usar Java Reflection tendremos que sacrificar algo y ese algo es el tema de rendimiento. El código se ejecutará bastante más despacio que un simple código con bucles . Pero en mas de una ocasión nos puede salvar de muchas horas de desarrollo.

Acabamos de ver como usar Java equals y hashcode

Otros artículos relacionados:

1. Java Override y encapsulación
2. Herencia y relaciones entre objetos
3. Java Generics (II) uso de WildCard