



JAVA REFLECTION

Emiliano Loya Flores

Programación Orientada a Objetos

Java Reflection

Es la capacidad que nos da Java para permitir a los programas inspeccionar y modificar la estructura interna de las clases, interfaces, campos y métodos de un programa en tiempo de ejecución.

Esta API es comúnmente utilizada en:

- IDEs.
- Debuggers.
- Herramientas de pruebas.

Aplicaciones

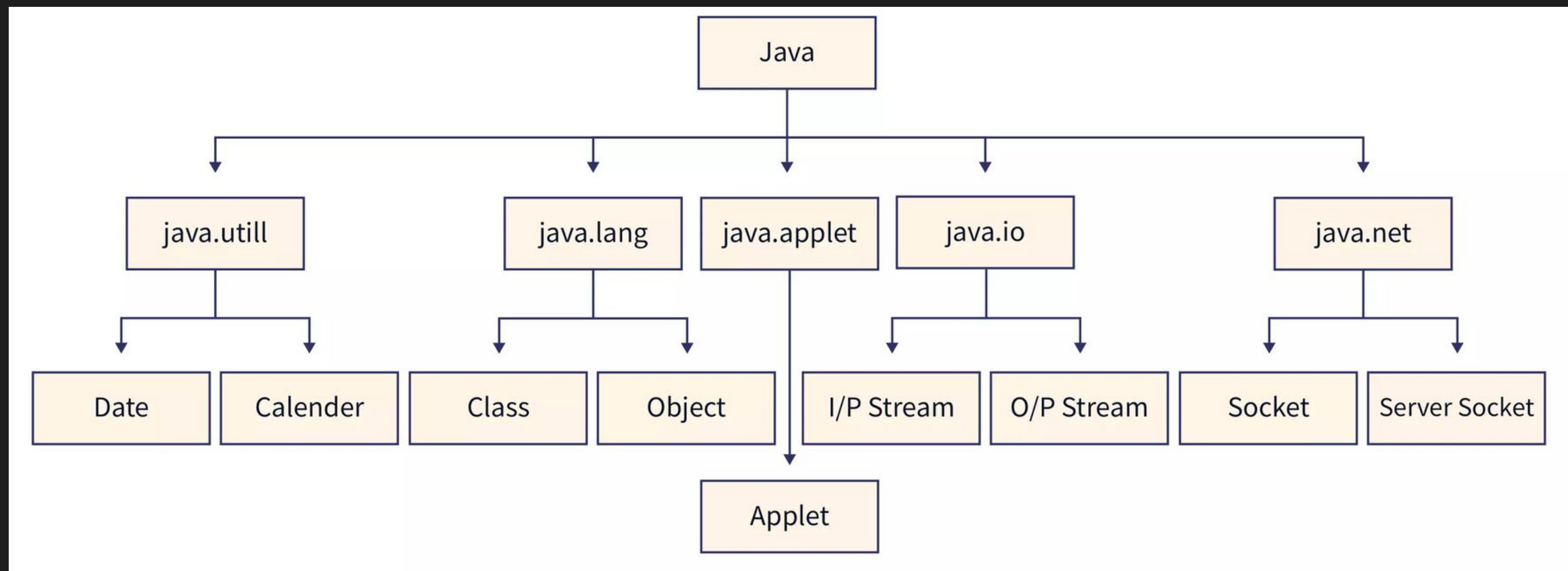
Entre las aplicaciones más comunes de Java reflection se encuentran:

- La carga dinámica de clases.
- La invocación dinámica de métodos.
- La creación de objetos en tiempo de ejecución.
- La inspección de metadatos de clase.

Clase: java.lang.Class

Esta clase tiene esencialmente dos usos:

- Da métodos para obtener la metadata de una clase en tiempo de ejecución.
- Da métodos para examinar y cambiar el comportamiento de una clase en tiempo de ejecución.





Métodos más comunes de java.lang.Class

Método	Descripción
<code>public String getName()</code>	Regresa el nombre de la clase.
<code>public static Class.forName(String className)throws ClassNotFoundException</code>	Carga la clase y regresa su referencia.
<code>public Object newInstance()throws InstantiationException,IllegalAccessException</code>	Crea una nueva instancia.
<code>public boolean isInterface()</code>	Comprueba si es una interfaz.
<code>public boolean isArray()</code>	Comprueba si es un arreglo.



<code>public boolean isPrimitive()</code>	Comprueba si es primitivo.
<code>public Class getSuperclass()</code>	Regresa la referencia de la superclase.
<code>public Field[] getDeclaredFields()throws SecurityException</code>	Regresa el número total de campos de la clase.
<code>public Method[] getDeclaredMethods()throws SecurityException</code>	Regresa el número total de métodos de la clase.
<code>public boolean isArray()public Constructor[] getDeclaredConstructors()throws SecurityException</code>	Regresa el número total de constructores de la clase.
<code>public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException</code>	Regresa una instancia del método de la clase.

Obtener el objeto de la clase Class:

Método forName()

- Se usa para cargar la clase dinámicamente.
- Regresa la instancia de la clase Class.
- Se usa cuando se conoce el nombre de la clase.
- No se puede usar para tipos primitivos.

```
class Simple {  
    public Simple() {  
    }  
}  
  
public class Test {  
    public static void main(String args[]) throws Exception {  
        Class c = Class.forName("Simple");  
        System.out.println(c.getName());  
    }  
}
```

Obtener el objeto de la clase Class:

Método getClass()

- Devuelve una instancia de la clase.
- Se usa cuando se conoce el tipo.
- Puede usarse con primitivos.

```
class Simple {  
  
    public class Test {  
        public static void main(String args[]) throws Exception  
        {  
            Simple s = new Simple();  
            Class c = s.getClass();  
            System.out.println(c.getName());  
        }  
    }  
}
```


Obtener el objeto de la clase Class:

Sintaxis .class

- Se usa cuando un tipo esta disponible pero no hay una instancia.
- Puede usarse con primitivos.

```
class Simple {  
  
}  
  
public class Test {  
    public static void main(String args[]) throws Exception  
    {  
        Class c = Simple.class;  
        System.out.println(c.getName());  
    }  
}
```

Analizar el objeto de la clase:

Método isInterface():

- Determina si el objeto de la clase es una interfaz.

Método isArray():

- Determina si el objeto de la clase es un arreglo.

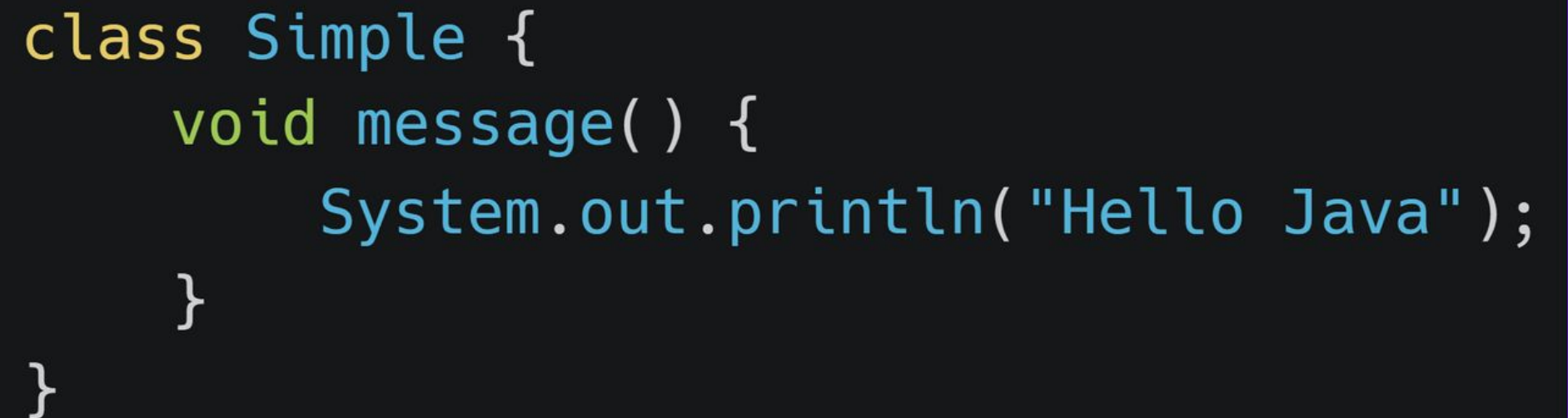
Método isPrimitive():

- Determina si el objeto de la clase es un primitivo.

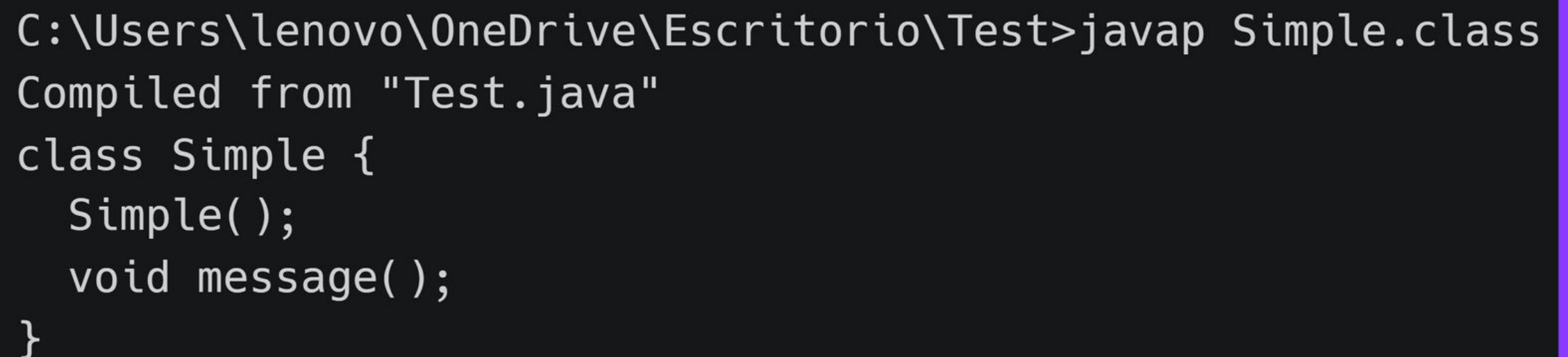
```
class Simple {  
  
}  
  
public class Test {  
    public static void main(String args[]) throws Exception {  
        Class c1 = Simple.class;  
        Simple[] c2 = new Simple[2];  
        Class k = int.class;  
  
        System.out.println(c1.isInterface());  
        System.out.println(c2.getClass().isArray());  
        System.out.println(k.isPrimitive());  
    }  
}
```

Herramienta javap

Es un comando que desensambla los archivos de tipo clase y muestra información sobre los campos, constructores y métodos que tiene dicha clase.



```
class Simple {  
    void message() {  
        System.out.println("Hello Java");  
    }  
}
```



```
C:\Users\lenovo\OneDrive\Escritorio\Test>javap Simple.class  
Compiled from "Test.java"  
class Simple {  
    Simple();  
    void message();  
}
```



Opciones de la herramienta javap

Opción	Descripción
-help	Imprime el mensaje de ayuda.
-l	Imprime el numero de linea y la variable local.
-c	Desensambla en código.
-s	Imprime la firma del tipo interno.
-sysinfo	Muestra la información del sistema (path, tamaño, fecha, MD5, hash)
-constants	Muestra las constantes.
-version	Muestra información de la versión

Acceder a métodos privados de una clase

Es posible acceder a métodos privados cambiando el comportamiento de una clase en tiempo de ejecución. Para esto son necesarios estos métodos:

```
public Method getDeclaredMethod(String name, Class[] parameterTypes) throws  
NoSuchMethodException, SecurityException
```

- Devuelve un objeto de tipo Método, que refleja el método de la clase que queremos usar.

```
public void setAccessible(boolean status) throws SecurityException
```

- Establece la accesibilidad del método.

```
public Object invoke(Object method, Object... args) throws IllegalAccessException,  
IllegalArgumentException, InvocationTargetException
```

- Se usa para invocar el método.



Acceder a métodos privados de una clase sin parametros



```
import java.lang.reflect.Method;
class Simple {
    private void message() {
        System.out.println("Hello Java");
    }
}
public class Test {
    public static void main(String args[]) throws Exception {
        Class c = Class.forName("Simple");
        Object o = c.newInstance();
        Method m = c.getDeclaredMethod("message", null);
        m.setAccessible(true);
        m.invoke(o, null);
    }
}
```



Acceder a métodos privados de una clase con parametros

```
import java.lang.reflect.Method;
class Simple {
    private void message(int i) {
        System.out.println("This is your number: " + i);
    }
}
public class Test {
    public static void main(String args[]) throws Exception {
        Class c = Class.forName("Simple");
        Object o = c.newInstance();
        Method m = c.getDeclaredMethod("message", int.class);
        m.setAccessible(true);
        m.invoke(o, 10);
    }
}
```



Acceder a constructores privados de una clase

Así como los métodos, los constructores privados también son accesibles por nosotros. Es muy parecida la manera en que hacemos esto:

```
import java.lang.reflect.Constructor;

class Simple {
    private Simple() {
        System.out.println("Hola Mundo");
    }
}

public class Test {
    public static void main(String args[]) throws Exception
    {
        Constructor<Simple> simple =
            Simple.class.getDeclaredConstructor(null);
        simple.setAccessible(true);
        Object obj = simple.newInstance();
    }
}
```


Ejercicio:



Crea una clase llamada "Car", la cual tendrá lo siguiente:

Atributos (Privados):

- name: String
- year: int

Constructores (Ambos privados):

- constructor vacío que inicialice los atributos como:
 - name = ""
 - year = 0
- Car(name: String, year: int)

Método (Privado):

- void: getCarInfo(): El cual mostrará los valores de los atributos.

Método Main:

- Se analizará la clase Car con los tres métodos antes vistos, obteniendo el objeto de la clase con cualquier opción, y se mostrarán los resultados.
- Se creará un Car con el constructor vacío.