1 Definición y uso.

En programación de ordenadores:
- La reflexión es una parte del lenguaje que permite a un programa conocerse así mismo en tiempo de ejecución.
- Consiste en metadatos y operaciones que los manipulen. (Un metadato es un dato que contiene información sobre otros datos. Por ejemplo, una objeto que contiene información sobre su clase: cuál es el nombre de esa clase, qué atributos o métodos tiene, etc.)

- La reflexión puede usarse para:
  - Construir objetos y arrays.
  - Acceder y cambiar el valor de atributos de instancia o de clase.
  - Invocar métodos de instancia y de clase.
  - Consultar y modificar elementos de vectores, etc...

- La reflexión permite hacer esto sin necesidad de conocer el nombre de las clases en tiempo de compilación, es decir, mientras que escribimos el código. Por ejemplo, porque leas las clases que tienes que crear de un fichero. Esta información puede ser proporcionada en forma de datos en tiempo de ejecución (por ejemplo, en un String).

- Ejemplo

De un fichero de texto, leemos la cadena "Barco", que indica que hay que crear un objeto de esa clase. Mediante reflexión, el programa actuaría de la siguiente forma Se preguntaría ¿existe una clase 'Barco' a la que tenga acceso? Si es así, la cargará en memoria.

```
Class c = Class.forName("Barco");
Object obj = c.newInstance(); // Invocaría a un constructor de la clase para  crear un objeto Barco.
```

El método forName devuelve la instancia del objeto Barco, **Barco.class** que contiene toda la inforrmación de la definición sobre dicha clase, y que fue cargada por la maquina virtual al empezar el programa, si dicha clase no fue cargada, se intentaría cargar durante la ejecución.

Java proporciona dos formas de **obtener información sobre los tipos en tiempo de ejecución**

RTTI tradicional (upcasting, downcasting)

Cuando el tipo de un objeto está disponible en tiempo de compilación y ejecución.
En la práctica el programador dispone de esos tipos ya creados y puede hacer cast hacia los tipos que conoce.

Reflexión

Cuando el tipo de un objeto puede no estar disponible en tiempo de compilación y/o ejecución. Puede ser que no conozcamos las clases a las que convertir o simplemente que no disponemos de las clases en ese momento.

Donde se encuentra

- java.lang.Class // este no hace falta incluirlo porque por defecto ya esta java.lang.*;
- java.lang.reflect.* // este si hace falta incluirlo.

Listado de utilidades por clases.

java.lang.reflect.Array
- Proporciona métodos estáticos para crear y acceder dinámicamente a los arrays.

java.lang.reflect.Member
- Interfaz que refleja información sobre un miembro de una clase (atributo, método o constructor)

java.lang.reflect.Constructor
- Proporciona información y acceso sobre un método constructor

java.lang.reflect.Field
- Proporcionan información y acceso dinámico a un atributo de una clase.
- El atributo puede ser de clase (static) o de instancia.

java.lang.reflect.Method
- Proporciona información y acceso a un método.

java.lang.Class
- Representa clases e interfaces

java.lang.Package
- Proporciona información sobre un paquete.

java.lang.ClassLoader
- Clase abstracta. Proporciona servicios para cargar clases dinámicamente.

2. Objeto class

Toda clase que se carga en la JVM tiene asociado un objeto de tipo Class -> Corresponde a un fichero .class
El cargador de clases (java.lang.ClassLoader) es el responsable de encontrar y cargar una clase en la JVM.

Carga dinámica de clases: (carga de una clase durante al instanciar un objeto...
JVM comprueba si la clase y a ha sido cargada
Localiza y carga la clase en caso necesario.
Una vez cargada, se usa para instanciar el objeto

Un objeto Class contiene información sobre una clase:
Sus métodos
Sus campos
Su superclase
Los interfaces que implementa
Si es o no un array
...

Class.forName(String)
Obtiene el objeto Class correspondiente a una clase cuyo nombre se pasa como argumento.

```
try
{
        // busca y carga la clase B, si no estaba ya cargada
        Class c = Class.forName ("B");
        // c apuntaría un objeto Class que representa a la clase B
}catch (ClassNotFoundException e)
{
        // B no existe, es decir, no estÃi en el CLASSPATH
        e.printStackTrace ();
}
```

En el tema anterior, hemos visto otras formas de obtener el objeto Class y/o acceder a la informaciÃ³n que contiene:
- Literales de clase

```
Circulo.class
Integer.class
int.class

Figura2D f = new Circulo();
// compara referencias, hay un objeto class por cada tipo de datos cargado por la maquina virtual.
if (f.getClass() == Circulo.class) // cierto
{ Circulo c = (Circulo)a; }
```

- instanceOf

```
if (x instanceof Circulo)
        ((Circulo) x).setRadio(10);
```

3. Factoria de objetos.

Con código condicional sin realizar reflexión:

```
public static Figura2D crearFigura(String s) throws ExcepcionFiguraDesconocida {
        Figura2D f = null;
        if(s.equals("Circulo"))
                f = new Circulo();
        else
                if(s.equals("Cuadrado"))
                        f = new Cuadrado();
                else
                        throw new ExcepcionFiguraDesconocida();
}
```

Usando reflexión eliminando el código condicional.

```
public static Figura2D crearFiguraReflexion(String descripcion){
        Figura2D f = null;
        Class c = null;
        try{
                // c apuntaría un objeto de tipo class que representa la clase indicada
                c = Class.forName(descripcion);
                f = (Figura2D) c.newInstance();
        } catch(ClassNotFoundException e){
                // si la clase indicada en descripcion no se puede cargar porque no está disponible.
        } catch (InstantiationException e) {
                // esta se lanza si no se puede crear la instancia, por ejemplo porque es una clase
                // abstracta o porque no tiene constructor por defecto.
        } catch (IllegalAccessException e) {
                // intenta crear una instacia de una clase a la cual no tiene acceso,
                // el objeto class puede llegar a un método de una clase que no vea dicha declaración.
        }
        return f;
}
```

4. Métodos de class

public String getName( );
- Devuelve el nombre de la clase reflejada por este objeto Class.

public boolean isInterface( );
- Devuelve cierto si la clase es un interfaz.

public boolean isArray( );
- Devuelve cierto si la clase es un array.

public Class getSuperclass( );
- Devuelve el objeto Class que representa a la clase base de la clase actual.

public Class[] getInterfaces( );
- Devuelve un array de objetos Class que representan los interfaces implementados por esta clase.

public Object newInstance( );
- Crea una instancia de esta clase (invocando al constructor por defecto)

public Constructor[] getConstructors( );
- Devuelve un array con todos los constructores públicos de la clase actual.

public Method[] getDeclaredMethods( );
- Devuelve un array de todos los métodos públicos y privados declarados en la clase actual.

public Method[] getMethods( );
- Devuelve un array de todos los métodos públicos en la clase actual, así como los declarados en todas las clases base o interfaces implementados por esta clase.

public Method getMethod(String methodName, Class[] paramTypes);
- Devuelve un objeto Method que representa el método público identificado por su nombre y tipo de parámetros, declarado en esta clase o heredado de una clase base.

public Method getDeclaredMethod(String methodName, Class[] paramTypes);
- Devuelve un objeto Method que representa el método público identificado por su nombre y tipo de parámetros, declarado en esta clase. El método puede ser privado.

5. La clase array.

```
public class Array {
        // todo mÃ©todos static y public:
        int getLength( Object arr );
        Object newInstance( Class elements, int length );
        Object get( Object arr, int index );
        void set( Object arr, int index, Object val );
        // Varias versiones especializadas, como...
        int getInt( Object arr, int index );
        void setInt( Object arr, int index, int val );
}

EJEMPLO TRIVIAL DE USO DE ARRAY.
Perro[] perrera = new Perro[10];  // Class c1 = Class.forName("Perro");
                                  // Perro [] perrera = (Perro[]) Array.newInstance(c1, 10);

// Longitud de un array.
int n = Array.getLength( perrera ); // int n = perrera.length();

// Asigna uno de los elementos del array
Array.set( perrera, (n-1), new Perro( "Spaniel" ) ); // perrera[n-1] = new Perro("Spaniel");

// Obtiene un objeto del array, determina su clase y muestra su valor:
Object obj = Array.get( perrera, (n-1) );
Class c1 = obj.getClass( );
System.out.println( c1.getName( ) + "-->" + obj.toString( ) );
```

Formas de declarar un array.

```
// 1:
        Perro perrera = new Perro[10];
// 2:
        Class c1 = Class.forName( "Perro" );
        Perro perrera = (Perro[]) Array.newInstance( c1, 10 )
```

6. Duplicar un vector.

CONOCIMIENTOS PREVIOS.

```
public static void main(String []args){
	// Un vector Object [] deja que apuntes a un String [], porque Object es un derivado de String.
	String [] vs = new String []{"hola", "adios"};
	Object [] vo = vs;
	// La conversion aqui es totalmente valida.
	String [] vs2 = (String []) vo;
	for(int i = 0; i < vs2.length; i++){
		System.out.println(vs2[i]);
	}

	// Esta no es la misma situacion que la anterior es un vector de objects donde
	// cada elemento apunta a un String, no apunta a un vector de String si no a
	// un vector de Object que cada elemento apunta a un String, conclusion, para que
	// se pueda realizar la conversiÃ³n, no vale con que los elementos sean todos de un tipo
	// ademas, no tendria sentido.
	Object [] vobj = new Object[]{"pepe", "juan", "luis"};
	String [] vcads = (String []) vobj;
	for(int i = 0; i < vcads.length; i++)
		System.out.println(vcads[i]);
	}
}
```

DUPLICAR UN ARRAY.

```
public static Object[] doubleArrayBad( Object[] arr)
{
	int newSize = arr.length * 2 + 1;
	Object[] newArray = new Object[ newSize ];
	for( int i = 0; i < arr.length; i++ )
		newArray[ i ] = arr[ i ];
}

// Que entra si hacemos?
String []cads = new String [] {"hola", "adios"};
String []cadsx2 = doubleArrayBad(cads);
// esto no devolveria un Object [] apuntando a un String []
// si no un Object [] apuntado a String cabroncetes.


// TENEMOS QUE SOLUCIONARLO REFLEXIONANDO
public static Object[] doubleArray(Object [] arr){
	int newSize = arr.length * 2 + 1;
	Class c1 = arr[0].getClass();
	Object [] newArray = Array.newInstance(c1, newSize);
	Object[] newArray = (Object[]) Array.newInstance(c1.getComponentType( ), newSize );
		for(int i = 0; i < arr.length; i++)
			newArray[i] = arr[i];
		return newArray;
	}
```