Retrieving Class instances

The java.lang.Class takes a central place in the reflection package. It represents a structure of classes and interfaces by aggregating information on constructors, fields, methods, superclasses, interfaces, and so on.

If you have an instance of a class of a specific type, you can obtain all information about the type.

The .class Syntax

To retrieve a Class instance for a given type we can use .class construction:
Class stringClass = String.class;

This way of obtaining a Class object is useful if we don't have any instances of the class available.

This is also the easiest way to obtain the Class for a primitive type and even void:

Class intClass = int.class;
Class voidClass = void.class;

Retrieve Class from an object instance

The class Object, the base class for any reference type, has a getClass method. To get Class of a given instance, it's enough to call this method:

Class instanceClass = "abc".getClass();
Don't forget that this only works for reference types since they inherit from Object! For primitive types, you might want to use other methods.

Retrieve Class with a given name

If we have access to a fully qualified type name, we can obtain the corresponding Class using the static method Class.forName. **Keep in mind that this method cannot be used for primitive types!**
Class forName = Class.forName("java.lang.String");

This method can also be used to retrieve Class objects for array classes. In this case, the name consists of the name of the element type preceded by one or more **[** characters representing the depth of the array nesting. The element types are encoded in the
following way:

- boolean – Z
- byte – B
- char – C

- class or interface – L*classname;*
- double – D
- float – F
- int – I
- long – J
- short – S

Class floatArrayClass = Class.forName("[F");
Class objectArrayClass = Class.forName("[[Ljava.lang.Object;");
Class scannerArrayClass = Class.forName("[Ljava.util.Scanner;");

The variable floatArrayClass will contain the Class corresponding to a one-dimensional array of primitive type float (the same as float[].class). The variable objectArrayClass in its turn will contain the Class corresponding to a two-dimensional array of Object

Note, that there should be a semicolon ; after an array of any objects.

Methods that Return Classes

In addition to the methods we've described above, we can use some Reflection APIs to get classes.

// Returns the super class for the given class
String.class.getSuperclass();

// Returns all the public classes, interfaces, and enums that are members of the class
String.class.getClasses();

// Returns all of the classes, interfaces, and enums that are explicitly declared in this class.
String.class.getDeclaredClasses();

Getting class by name

We have covered the two methods for obtaining a class by name. Let's sum up their pros and cons.
The first way is getting a class directly, for instance String.class. It looks simple but means that we're aware of a class at the compile time.
The second way is by using the method forName of Class, for instance Class.forName("java.lang.String"). This way works at runtime as well as it can be used when a target class name is resolved dynamically, for example, retrieved from a config.