Theory: Retrieving Class instances

24 minutes 0 / 5 problems solved Skip this topic Start practicing

1073 users solved this topic. Latest completion was about 15 hours ago.

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. In other words, if you have an instance of the class of a specific type, you can obtain all about the type. That's why `java.lang.Class` is so important. There are several ways to get an object representing a `java.lang.Class` depending on whether the code has access to an instance of the class, the name of the class, the type, or an existing `Class`. Let's take a closer look at them!

Current topic:

Retrieving Class instances

Topic depends on:

Reflection basics

Topic is required for:

Dealing with modifiers

Table of contents:

↑ Retrieving Class instances

§1. The .class Syntax

§2. Retrieve Class from an object instance

§3. Retrieve Class with a given name

§4. Methods that Return Classes

§5. Getting class by name

§6. Conclusion

Feedback & Comments

§1. The .class Syntax

To retrieve a `Class` instance for a given type we can use `.class` construction:

```
Class stringClass = String.class;
```

The `.class` is simply added to the name of the type. 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;
```

§2. 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.

§3. 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.

§4. Methods that Return Classes

In addition to the methods we've described above, we can use some Reflection APIs to get classes. However, you should keep in mind that they can be used only if a `Class` has already been obtained.

Here are some examples:

```
// 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();
```

§5. 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.

§6. Conclusion

`java.lang.Class` aggregates all the information that describes a given type. That is the reason why it is important to know ways of getting an instance of the class. If a class name is known, there are two ways of getting an instance of `java.lang.Class`: calling `ClassName.class` directly or getting `Class.forName("package.ClassName")`. The alternative ways are based on having a reference to a class. For example, if you have an object, you can obtain its class as well.

Report a typo

71 users liked this theory. 2 didn't like it. What about you?

😍 🙂 😐 🙁 😡

Start practicing

Comments (5) Hints (0) Useful links (0) Show discussion