(1)

Reflection Api Gateway & Wildcards :

(I) Introduction to Reflection Entry Point : (Gateway) :-

(a) Class<?> is the entry Point for us to reflect on app code.

Here '?' is a Java WildCard

(b) An object of Class<?> contains all info on
(i) The given object runtime (ii) Object Class in application code.
i.e What methods and field it contains, Params And their values,
What classes / Interfaces it extends etc.

(c) To get the object of Class<?>
(i) Object.getClass() :⇒ String str = "xyz" ; Class<String> c = str.getClass();
(ii) Suffix ".class" to a type name ⇒ Class<?> map1 = HashMap.class;
(iii) Class.forName("Fully specified ClassName") :

② -> The "Object.getClass()" to get Class<?> object doesnot work with primitive type But The ".class suffix" to get <Class<??> object work with Primitive types too.

Eg: int x = 5; Class c = x.getClass() (X) [Compilation error]
Class c = int.class (✓) [Valid].

This is important when we want to get Class<?> object for Params of functions

③ -> The Class.forName(-) approach is the least safest as the class is given as String parameter which may/may not exist and thus gives Runtime Exceptions (ClassNotFoundException).

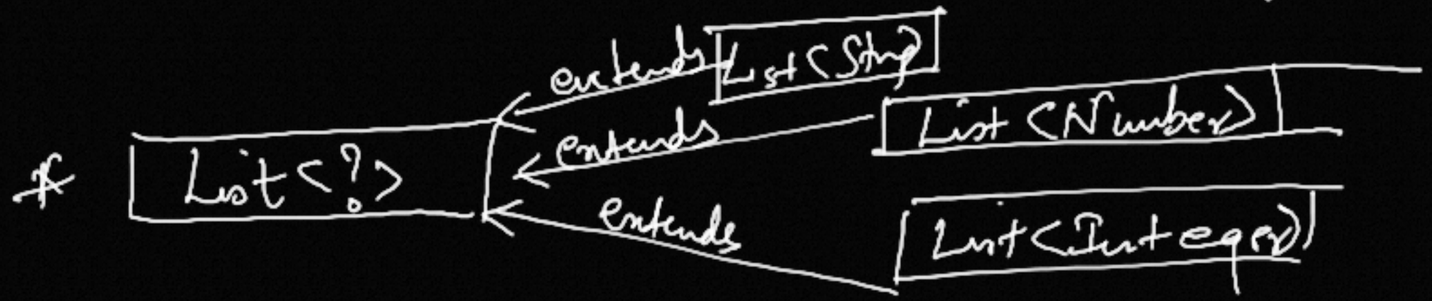
This approach is generally used when ClassType is unknown to appcode and is passed using some config file at runtime.

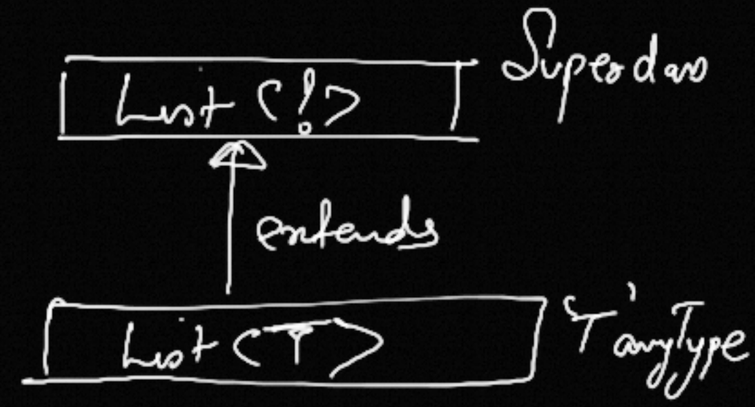
Like in JDBC, the DriverManager ClassType String is a dependency & not in your appcode.

③ (d) Now lets talk about '?' in Class<?> : [JAVA WILDCARDS] [Generics]

Although : Number ← extends — Integer ;

But List<Number> ← ✗ extends — List<Integer>



* 

This type of relationship is True for any class Types : i.e

* So Using Class<?> we can describe a class object of any Parameter type.

* Class<?> is Super type to class<T> of any type 'T'.

* Useful when Compiler Doesn't know Generic Type at Compile time eg:

④ eg: Class<?> carClass = Class.forName("vehicles.Car");

* Also useful when class has Generic Parameters i.e. Class is Generic itself.
OR When you have something like DMD:

Also has DMD.

Class is Generic in itself

Eg: Map<String, String> genericMap = new HashMap<>();

Class<?> class = genericMap.getClass();

Will correspond to 'HashMap' and not 'Map'

* Also like in Generics 'T': using '?' we can also make Restrictions on extends

eg: Class<? extends Collection> clazz

Do '?' Here can correspond to 'List', 'Set' etc. only.

[END ON CLASS 1].