PIC 20A
class Class, Type Introspection, and Reflection

David Hyde
UCLA Mathematics

Last edited: June 3, 2020

class Class

The `class` `Object` has the method

```java
public Class<?> getClass()
```

The return type is `java.lang.Class<?>`.

Remember that `Class<?>` is a superclass of `Class<T>` for any type `T`.

The method signature says `T.getClass()` returns a `Class<?>`.
Specifically, it returns a `Class<T>`.

class Class

Find out what `class` your object is, with `getClass()` and `toString()`.

```
String s = "";
Class<?> c1 = s.getClass();
System.out.println(c1.getClass());
//output class java.lang.String
```

This is a useful debugging device.

Class<?> to Class<T>

```
String s = "";
//Class<String> c2 = s.getClass(); //error
Class<String> c2 = (Class<String>) s.getClass();
```

This is a narrowing reference conversion from the parent class `Class<?>` to the child class `Class<String>`.

Type introspection and reflection

In programming, *type introspection* is examining the type or properties of an object at runtime.

Reflection, which goes one step further, is manipulating the values, meta-data, properties and/or functions of an object at runtime.

C++ supports only type introspection and not reflection.
Java supports both type introspection and reflection.

Type introspection and reflection: example

Write a class with a method called printField().

```java
class SomeClass {
  public int field;
  public SomeClass(int field) {
    this.field = field;
  }
  public void printField() {
    System.out.println(field);
  }
}
```

Type introspection and reflection: example

You can do “if a method named printField exists, call the method named printField”.

```
public class Test {
  public static void main(String[] args)
                                      throws Exception {
    SomeClass obj = new SomeClass(4);
    Class<?> c = obj.getClass();
    try {
      Method m = c.getMethod("printField");
      m.invoke(obj); //output 4
    } catch (NoSuchMethodException e) {
      System.out.println(
        "SomeClass does not have that method.");
    }
  }
}
```

Type introspection and reflection: example

Checking whether a method named `printField` exists is type introspection.

```
try {
  c.getMethod("printField");
} catch (NoSuchMethodException e) {
  System.out.println(
        "SomeClass does not have that method.");
}
```

Calling a method named `printField` exists is reflection.

```
m.invoke(obj);
```

Conclusion

Type introspection is a very useful debugging tool.

As a tool for production code, type introspection and reflection are advanced tools that are rarely helpful.

Type introspection and reflection are interesting, but you should rarely use them outside of debugging.