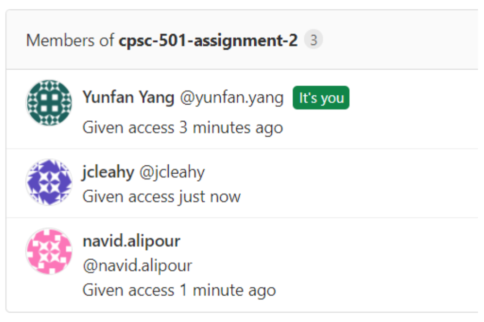
Yunfan Yang 30067857

https://gitlab.cpsc.ucalgary.ca/yunfan.yang/cpsc-501-assignment-3/

The access to this repository has been granted to the TA and the professor. Please let me know if it is not working by sending email to yunfan.yang1@ucalgary.ca so I can fix. 😊



Members of **cpsc-501-assignment-2** 3

Yunfan Yang @yunfan.yang It's you
Given access 3 minutes ago

jcleahy @jcleahy
Given access just now

navid.alipour
@navid.alipour
Given access 1 minute ago

The bonus part is implemented: `DriverBonus.java`

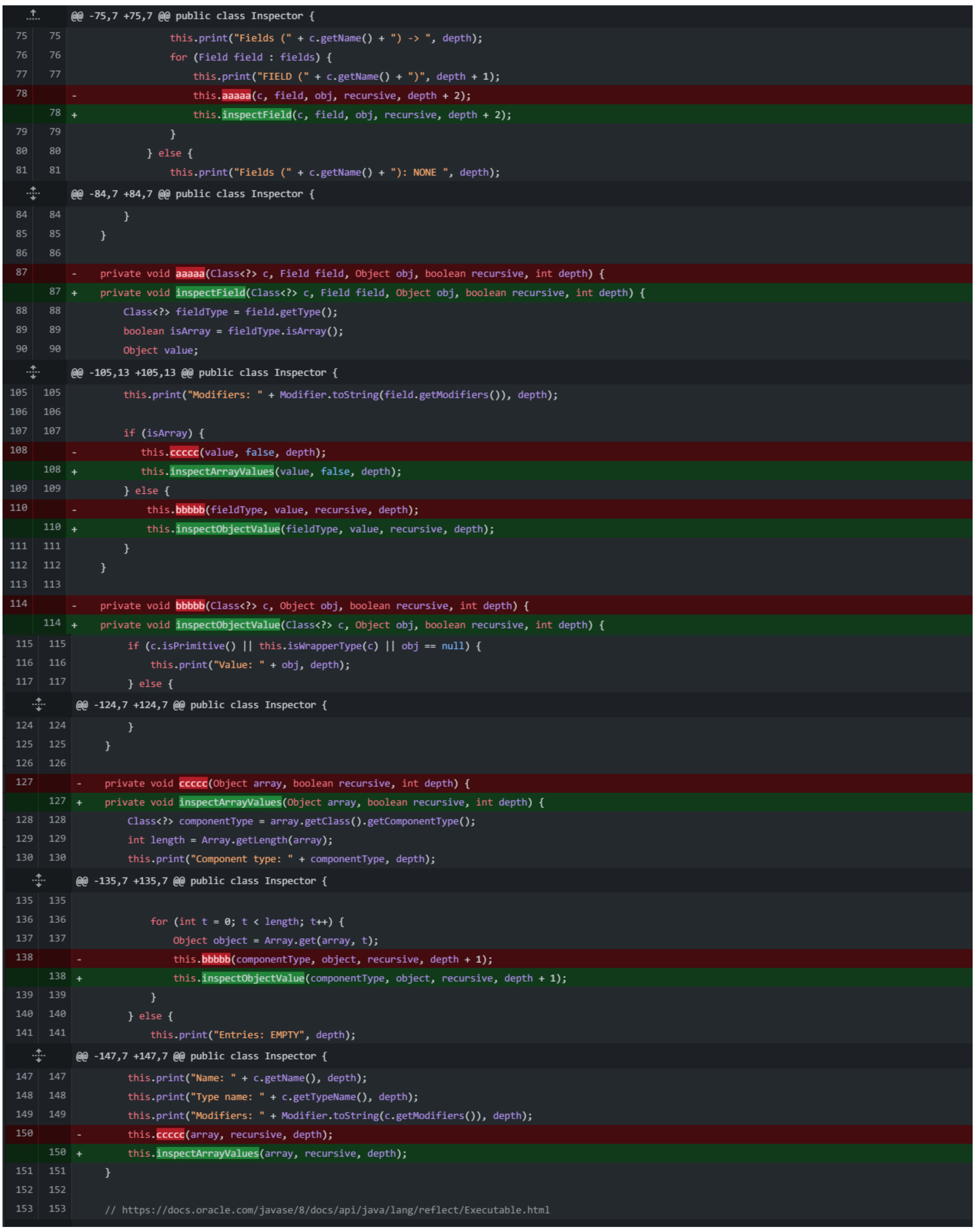
(Refactorings start on the next page)

Refactorings

Rename Method

`Commit: aff2c780342f21136becc96347e64e95cc4b2718`

Some method names are not descriptive at all at the time of coding because of laziness, for example, aaaaa or bbbbb. In order to improve the readability of the code, apply this refactoring to rename these methods with more descriptive names.

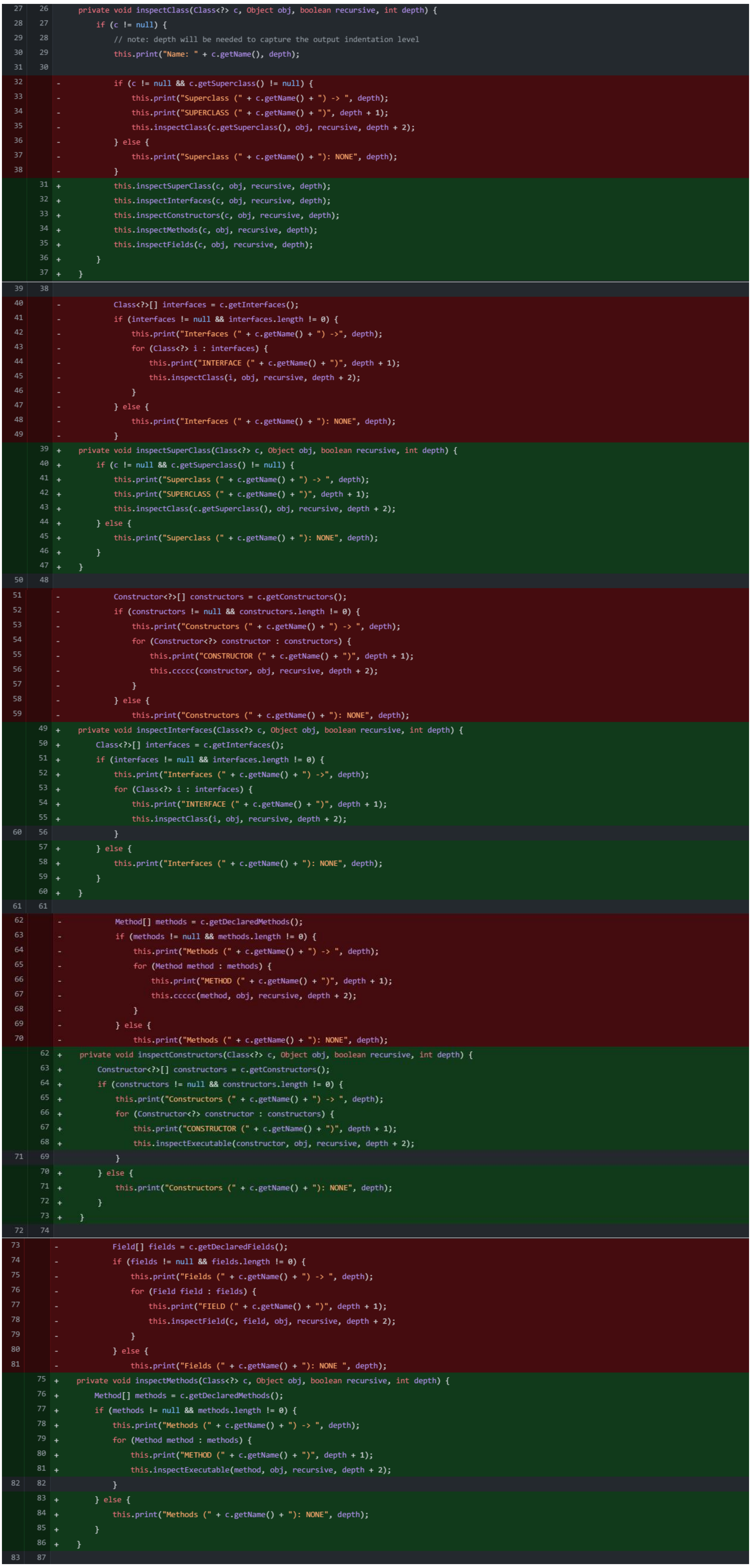


```diff
@@ -75,7 +75,7 @@ public class Inspector {
                 this.print("Fields (" + c.getName() + ") -> ", depth);
                 for (Field field : fields) {
                     this.print("FIELD (" + c.getName() + ")", depth + 1);
-                    this.aaaaa(c, field, obj, recursive, depth + 2);
+                    this.inspectField(c, field, obj, recursive, depth + 2);
                 }
             } else {
                 this.print("Fields (" + c.getName() + "): NONE ", depth);
@@ -84,7 +84,7 @@ public class Inspector {
         }
     }

-    private void aaaaa(Class<?> c, Field field, Object obj, boolean recursive, int depth) {
+    private void inspectField(Class<?> c, Field field, Object obj, boolean recursive, int depth) {
         Class<?> fieldType = field.getType();
         boolean isArray = fieldType.isArray();
         Object value;
@@ -105,13 +105,13 @@ public class Inspector {
         this.print("Modifiers: " + Modifier.toString(field.getModifiers()), depth);

         if (isArray) {
-            this.ccccc(value, false, depth);
+            this.inspectArrayValues(value, false, depth);
         } else {
-            this.bbbbb(fieldType, value, recursive, depth);
+            this.inspectObjectValue(fieldType, value, recursive, depth);
         }
     }

-    private void bbbbb(Class<?> c, Object obj, boolean recursive, int depth) {
+    private void inspectObjectValue(Class<?> c, Object obj, boolean recursive, int depth) {
         if (c.isPrimitive() || this.isWrapperType(c) || obj == null) {
             this.print("Value: " + obj, depth);
         } else {
@@ -124,7 +124,7 @@ public class Inspector {
         }
     }

-    private void ccccc(Object array, boolean recursive, int depth) {
+    private void inspectArrayValues(Object array, boolean recursive, int depth) {
         Class<?> componentType = array.getClass().getComponentType();
         int length = Array.getLength(array);
         this.print("Component type: " + componentType, depth);
@@ -135,7 +135,7 @@ public class Inspector {

             for (int t = 0; t < length; t++) {
                 Object object = Array.get(array, t);
-                this.bbbbb(componentType, object, recursive, depth + 1);
+                this.inspectObjectValue(componentType, object, recursive, depth + 1);
             }
         } else {
             this.print("Entries: EMPTY", depth);
@@ -147,7 +147,7 @@ public class Inspector {
         this.print("Name: " + c.getName(), depth);
         this.print("Type name: " + c.getTypeName(), depth);
         this.print("Modifiers: " + Modifier.toString(c.getModifiers()), depth);
-        this.ccccc(array, recursive, depth);
+        this.inspectArrayValues(array, recursive, depth);
     }

     // https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Executable.html
```

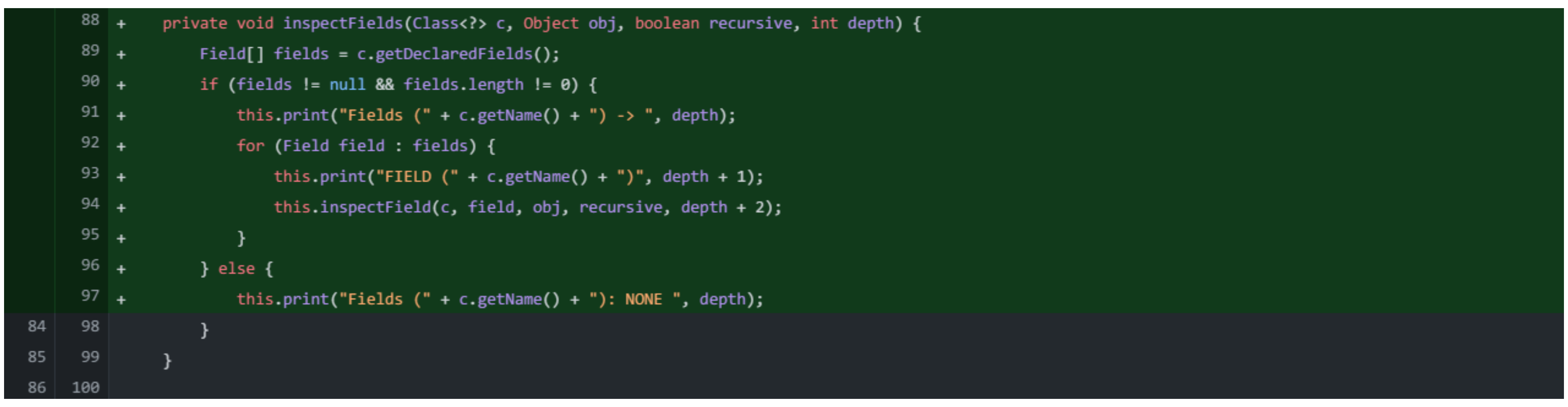
Extract Method

`Commit: 3534e67291e0ffaab0abd53b4294105b29549581`

The original version of inspectClass method is really long, it does too many things and there are more than 60 lines. By applying this refactoring, smaller methods are created and shorten the length of the method, which improves readability and the philosophy of OOP.



```diff
     private void inspectClass(Class<?> c, Object obj, boolean recursive, int depth) {
         if (c != null) {
             // note: depth will be needed to capture the output indentation level
             this.print("Name: " + c.getName(), depth);

-            if (c != null && c.getSuperclass() != null) {
-                this.print("Superclass (" + c.getName() + ") -> ", depth);
-                this.print("SUPERCLASS (" + c.getName() + ")", depth + 1);
-                this.inspectClass(c.getSuperclass(), obj, recursive, depth + 2);
-            } else {
-                this.print("Superclass (" + c.getName() + "): NONE", depth);
-            }
+            this.inspectSuperClass(c, obj, recursive, depth);
+            this.inspectInterfaces(c, obj, recursive, depth);
+            this.inspectConstructors(c, obj, recursive, depth);
+            this.inspectMethods(c, obj, recursive, depth);
+            this.inspectFields(c, obj, recursive, depth);
+        }
+    }

-            Class<?>[] interfaces = c.getInterfaces();
-            if (interfaces != null && interfaces.length != 0) {
-                this.print("Interfaces (" + c.getName() + ") ->", depth);
-                for (Class<?> i : interfaces) {
-                    this.print("INTERFACE (" + c.getName() + ")", depth + 1);
-                    this.inspectClass(i, obj, recursive, depth + 2);
-                }
-            } else {
-                this.print("Interfaces (" + c.getName() + "): NONE", depth);
-            }
+    private void inspectSuperClass(Class<?> c, Object obj, boolean recursive, int depth) {
+        if (c != null && c.getSuperclass() != null) {
+            this.print("Superclass (" + c.getName() + ") -> ", depth);
+            this.print("SUPERCLASS (" + c.getName() + ")", depth + 1);
+            this.inspectClass(c.getSuperclass(), obj, recursive, depth + 2);
+        } else {
+            this.print("Superclass (" + c.getName() + "): NONE", depth);
+        }
+    }

-            Constructor<?>[] constructors = c.getConstructors();
-            if (constructors != null && constructors.length != 0) {
-                this.print("Constructors (" + c.getName() + ") -> ", depth);
-                for (Constructor<?> constructor : constructors) {
-                    this.print("CONSTRUCTOR (" + c.getName() + ")", depth + 1);
-                    this.ccccc(constructor, obj, recursive, depth + 2);
-                }
-            } else {
-                this.print("Constructors (" + c.getName() + "): NONE", depth);
+    private void inspectInterfaces(Class<?> c, Object obj, boolean recursive, int depth) {
+        Class<?>[] interfaces = c.getInterfaces();
+        if (interfaces != null && interfaces.length != 0) {
+            this.print("Interfaces (" + c.getName() + ") ->", depth);
+            for (Class<?> i : interfaces) {
+                this.print("INTERFACE (" + c.getName() + ")", depth + 1);
+                this.inspectClass(i, obj, recursive, depth + 2);
             }
+        } else {
+            this.print("Interfaces (" + c.getName() + "): NONE", depth);
+        }
+    }

-            Method[] methods = c.getDeclaredMethods();
-            if (methods != null && methods.length != 0) {
-                this.print("Methods (" + c.getName() + ") -> ", depth);
-                for (Method method : methods) {
-                    this.print("METHOD (" + c.getName() + ")", depth + 1);
-                    this.ccccc(method, obj, recursive, depth + 2);
-                }
-            } else {
-                this.print("Methods (" + c.getName() + "): NONE", depth);
+    private void inspectConstructors(Class<?> c, Object obj, boolean recursive, int depth) {
+        Constructor<?>[] constructors = c.getConstructors();
+        if (constructors != null && constructors.length != 0) {
+            this.print("Constructors (" + c.getName() + ") -> ", depth);
+            for (Constructor<?> constructor : constructors) {
+                this.print("CONSTRUCTOR (" + c.getName() + ")", depth + 1);
+                this.inspectExecutable(constructor, obj, recursive, depth + 2);
             }
+        } else {
+            this.print("Constructors (" + c.getName() + "): NONE", depth);
+        }
+    }

-            Field[] fields = c.getDeclaredFields();
-            if (fields != null && fields.length != 0) {
-                this.print("Fields (" + c.getName() + ") -> ", depth);
-                for (Field field : fields) {
-                    this.print("FIELD (" + c.getName() + ")", depth + 1);
-                    this.inspectField(c, field, obj, recursive, depth + 2);
-                }
-            } else {
-                this.print("Fields (" + c.getName() + "): NONE ", depth);
+    private void inspectMethods(Class<?> c, Object obj, boolean recursive, int depth) {
+        Method[] methods = c.getDeclaredMethods();
+        if (methods != null && methods.length != 0) {
+            this.print("Methods (" + c.getName() + ") -> ", depth);
+            for (Method method : methods) {
+                this.print("METHOD (" + c.getName() + ")", depth + 1);
+                this.inspectExecutable(method, obj, recursive, depth + 2);
             }
+        } else {
+            this.print("Methods (" + c.getName() + "): NONE", depth);
+        }
+    }

```



```
+    private void inspectFields(Class<?> c, Object obj, boolean recursive, int depth) {
+        Field[] fields = c.getDeclaredFields();
+        if (fields != null && fields.length != 0) {
+            this.print("Fields (" + c.getName() + ") -> ", depth);
+            for (Field field : fields) {
+                this.print("FIELD (" + c.getName() + ")", depth + 1);
+                this.inspectField(c, field, obj, recursive, depth + 2);
+            }
+        } else {
+            this.print("Fields (" + c.getName() + "): NONE ", depth);
         }
     }

```