


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

Given access 3 minutes ago


It's you



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.

```

  ↑...  @@ -75,7 +75,7 @@ public class Inspector {
75 75      this.print("Fields (" + c.getName() + ") -> ", depth);
76 76      for (Field field : fields) {
77 77          this.print("FIELD (" + c.getName() + ")", depth + 1);
78 -      this.aaaaa(c, field, obj, recursive, depth + 2);
+      this.inspectField(c, field, obj, recursive, depth + 2);
79 79      }
80 80      } else {
81 81          this.print("Fields (" + c.getName() + "): NONE ", depth);
  ⚡@@ -84,7 +84,7 @@ public class Inspector {
84 84      }
85 85      }
86 86
87 - 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) {
88 88     Class<?> fieldType = field.getType();
89 89     boolean isArray = fieldType.isArray();
90 90     Object value;
  ⚡@@ -105,13 +105,13 @@ public class Inspector {
105 105     this.print("Modifiers: " + Modifier.toString(field.getModifiers()), depth);
106 106
107 107     if (isArray) {
108 -         this.ccccc(value, false, depth);
+         this.inspectArrayValues(value, false, depth);
109 109     } else {
110 -         this.bbbbb(fieldType, value, recursive, depth);
+         this.inspectObjectValue(fieldType, value, recursive, depth);
111 111     }
112 112 }
113 113
114 - private void bbbbb(Class<?> c, Object obj, boolean recursive, int depth) {
+ private void inspectObjectValue(Class<?> c, Object obj, boolean recursive, int depth) {
115 115     if (c.isPrimitive() || this.isWrapperType(c) || obj == null) {
116 116         this.print("Value: " + obj, depth);
117 117     } else {
  ⚡@@ -124,7 +124,7 @@ public class Inspector {
124 124     }
125 125     }
126 126
127 - private void ccccc(Object array, boolean recursive, int depth) {
+ private void inspectArrayValues(Object array, boolean recursive, int depth) {
128 128     Class<?> componentType = array.getClass().getComponentType();
129 129     int length = Array.getLength(array);
130 130     this.print("Component type: " + componentType, depth);
  ⚡@@ -135,7 +135,7 @@ public class Inspector {
135 135
136 136     for (int t = 0; t < length; t++) {
137 137         Object object = Array.get(array, t);
138 -         this.bbbbb(componentType, object, recursive, depth + 1);
+         this.inspectObjectValue(componentType, object, recursive, depth + 1);
139 139     }
140 140     } else {
141 141         this.print("Entries: EMPTY", depth);
  ⚡@@ -147,7 +147,7 @@ public class Inspector {
147 147     this.print("Name: " + c.getName(), depth);
148 148     this.print("Type name: " + c.getTypeName(), depth);
149 149     this.print("Modifiers: " + Modifier.toString(c.getModifiers()), depth);
150 -     this.ccccc(array, recursive, depth);
+     this.inspectArrayValues(array, recursive, depth);
151 151 }
152 152
153 153     // 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.

27	26	private void inspectClass(Class<?> c, Object obj, boolean recursive, int depth) {
28	27	if (c != null) {
29	28	// note: depth will be needed to capture the output indentation level
30	29	this.print("Name: " + c.getName(), depth);
31	30	
32	-	if (c != null && c.getSuperclass() != null) {
33	-	this.print("Superclass (" + c.getName() + ") -> ", depth);
34	-	this.print("SUPERCLASS (" + c.getName() + ")", depth + 1);
35	-	this.inspectClass(c.getSuperclass(), obj, recursive, depth + 2);
36	-	} else {
37	-	this.print("Superclass (" + c.getName() + "): NONE", depth);
38	-	}
	31 +	this.inspectSuperClass(c, obj, recursive, depth);
	32 +	this.inspectInterfaces(c, obj, recursive, depth);
	33 +	this.inspectConstructors(c, obj, recursive, depth);
	34 +	this.inspectMethods(c, obj, recursive, depth);
	35 +	this.inspectFields(c, obj, recursive, depth);
	36 +	}
	37 +	}
39	38	
40	-	Class<?>[] interfaces = c.getInterfaces();
41	-	if (interfaces != null && interfaces.length != 0) {
42	-	this.print("Interfaces (" + c.getName() + ") ->", depth);
43	-	for (Class<?> i : interfaces) {
44	-	this.print("INTERFACE (" + c.getName() + ")", depth + 1);
45	-	this.inspectClass(i, obj, recursive, depth + 2);
46	-	}
47	-	} else {
48	-	this.print("Interfaces (" + c.getName() + "): NONE", depth);
49	-	}
	39 +	private void inspectSuperClass(Class<?> c, Object obj, boolean recursive, int depth) {
	40 +	if (c != null && c.getSuperclass() != null) {
	41 +	this.print("Superclass (" + c.getName() + ") -> ", depth);
	42 +	this.print("SUPERCLASS (" + c.getName() + ")", depth + 1);
	43 +	this.inspectClass(c.getSuperclass(), obj, recursive, depth + 2);
	44 +	} else {
	45 +	this.print("Superclass (" + c.getName() + "): NONE", depth);
	46 +	}
	47 +	}
50	48	
51	-	Constructor<?>[] constructors = c.getConstructors();
52	-	if (constructors != null && constructors.length != 0) {
53	-	this.print("Constructors (" + c.getName() + ") -> ", depth);
54	-	for (Constructor<?> constructor : constructors) {
55	-	this.print("CONSTRUCTOR (" + c.getName() + ")", depth + 1);
56	-	this.ccccc(constructor, obj, recursive, depth + 2);
57	-	}
58	-	} else {
59	-	this.print("Constructors (" + c.getName() + "): NONE", depth);
	49 +	private void inspectInterfaces(Class<?> c, Object obj, boolean recursive, int depth) {
	50 +	Class<?>[] interfaces = c.getInterfaces();
	51 +	if (interfaces != null && interfaces.length != 0) {
	52 +	this.print("Interfaces (" + c.getName() + ") ->", depth);
	53 +	for (Class<?> i : interfaces) {
	54 +	this.print("INTERFACE (" + c.getName() + ")", depth + 1);
	55 +	this.inspectClass(i, obj, recursive, depth + 2);
60	56	}
	57 +	} else {
	58 +	this.print("Interfaces (" + c.getName() + "): NONE", depth);
	59 +	}
	60 +	}
61	61	
62	-	Method[] methods = c.getDeclaredMethods();
63	-	if (methods != null && methods.length != 0) {
64	-	this.print("Methods (" + c.getName() + ") -> ", depth);
65	-	for (Method method : methods) {
66	-	this.print("METHOD (" + c.getName() + ")", depth + 1);
67	-	this.ccccc(method, obj, recursive, depth + 2);
68	-	}
69	-	} else {
70	-	this.print("Methods (" + c.getName() + "): NONE", depth);
	62 +	private void inspectConstructors(Class<?> c, Object obj, boolean recursive, int depth) {
	63 +	Constructor<?>[] constructors = c.getConstructors();
	64 +	if (constructors != null && constructors.length != 0) {
	65 +	this.print("Constructors (" + c.getName() + ") -> ", depth);
	66 +	for (Constructor<?> constructor : constructors) {
	67 +	this.print("CONSTRUCTOR (" + c.getName() + ")", depth + 1);
	68 +	this.inspectExecutable(constructor, obj, recursive, depth + 2);
71	69	}
	70 +	} else {
	71 +	this.print("Constructors (" + c.getName() + "): NONE", depth);
	72 +	}
	73 +	}
72	74	
73	-	Field[] fields = c.getDeclaredFields();
74	-	if (fields != null && fields.length != 0) {
75	-	this.print("Fields (" + c.getName() + ") -> ", depth);
76	-	for (Field field : fields) {
77	-	this.print("FIELD (" + c.getName() + ")", depth + 1);
78	-	this.inspectField(c, field, obj, recursive, depth + 2);
79	-	}
80	-	} else {
81	-	this.print("Fields (" + c.getName() + "): NONE ", depth);
	75 +	private void inspectMethods(Class<?> c, Object obj, boolean recursive, int depth) {
	76 +	Method[] methods = c.getDeclaredMethods();
	77 +	if (methods != null && methods.length != 0) {
	78 +	this.print("Methods (" + c.getName() + ") -> ", depth);
	79 +	for (Method method : methods) {
	80 +	this.print("METHOD (" + c.getName() + ")", depth + 1);
	81 +	this.inspectExecutable(method, obj, recursive, depth + 2);
82	82	}
	83 +	} else {
	84 +	this.print("Methods (" + c.getName() + "): NONE", depth);
	85 +	}
	86 +	}
83	87	

```
88 + private void inspectFields(Class<?> c, Object obj, boolean recursive, int depth) {
89 +     Field[] fields = c.getDeclaredFields();
90 +     if (fields != null && fields.length != 0) {
91 +         this.print("Fields (" + c.getName() + ") -> ", depth);
92 +         for (Field field : fields) {
93 +             this.print("FIELD (" + c.getName() + ")", depth + 1);
94 +             this.inspectField(c, field, obj, recursive, depth + 2);
95 +         }
96 +     } else {
97 +         this.print("Fields (" + c.getName() + "): NONE ", depth);
84 98     }
85 99 }
86 100
```