CPSC501 – Assignment 2 – Version Control Logs

commit 0a44cfcfb1bd43832599ecc9d9c8774fa39a7d25

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-01 20:53:09

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-01 20:53:09

Parent: da9eaaa09c03a62365b3ab462f18a8282884b084 (Initial commit)

Child: 4a61a4292c39bc17728a47ef7675c92fd9d93da8 (Declaring, super and interface classes delt with)

Branches: master, origin/master

Initial creation

diff --git a/src/Inspector.java b/src/Inspector.java

new file mode 100644

index 0000000..88614f1

--- /dev/null

+++ b/src/Inspector.java

@@ -0,0 +1,7 @@

+

+public class Inspector {

+

+ public void inspect(Object obj, boolean recursive) {

+

+ }

+}

commit 4a61a4292c39bc17728a47ef7675c92fd9d93da8

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-01 21:28:06

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-01 21:28:06

Parent: 0a44cfcfb1bd43832599ecc9d9c8774fa39a7d25 (Initial creation)

Child: 1a911224b6d574a041705e48ad6284d0740bbe6f (inspects methods and constructors. Fields half complete.)

Branches: master, origin/master

Declaring, super and interface classes delt with

diff --git a/src/Inspector.java b/src/Inspector.java

index 88614f1..ca15749 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -1,7 +1,35 @@

+import java.lang.reflect.\*;

+import java.util.LinkedList;

+import java.util.List;

+

public class Inspector {

+ List<Class> toInspect = new LinkedList<Class>();

+ List<Class> alreadyInspected = new LinkedList<Class>();

public void inspect(Object obj, boolean recursive) {

+ String str = "";

+ Class classObj = obj.getClass();

+ alreadyInspected.add(classObj);

+ str += "Declaring Class: " + classObj.getName() + "\n";

+

+ Class superCl = classObj.getSuperclass();

+ if (superCl != null) {

+ str += "Superclass: " + superCl.getName() + "\n";

+ }

+ Class[] interfaceCl = classObj.getInterfaces();

+ for (int i = 0; i < interfaceCl.length; i++) {

+ str += "Interface: " + interfaceCl[i].getName() + "\n";

+ }

+

+

+

+

+

+

+

+

+ System.out.println(str);

}

}

commit 1a911224b6d574a041705e48ad6284d0740bbe6f

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-01 22:45:05

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-01 22:45:05

Parent: 4a61a4292c39bc17728a47ef7675c92fd9d93da8 (Declaring, super and interface classes delt with)

Child: c1ec9cce6313bb3c7173905e149102d326a0ef0c (Added recursive inspection for fields, superclass and interfaces)

Branches: master, origin/master

inspects methods and constructors. Fields half complete.

diff --git a/src/Inspector.java b/src/Inspector.java

index ca15749..a7f98a7 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -6,24 +6,133 @@

public class Inspector {

List<Class> toInspect = new LinkedList<Class>();

List<Class> alreadyInspected = new LinkedList<Class>();

-

+

public void inspect(Object obj, boolean recursive) {

+ inspectWorker(obj, recursive, 0);

+ }

+

+ public void inspectWorker(Object obj, boolean recursive, int ind) {

String str = "";

Class classObj = obj.getClass();

alreadyInspected.add(classObj);

- str += "Declaring Class: " + classObj.getName() + "\n";

+ str += indent(ind) + "Declaring Class: " + classObj.getName() + "\n";

Class superCl = classObj.getSuperclass();

if (superCl != null) {

- str += "Superclass: " + superCl.getName() + "\n";

+ str += indent(ind + 1) + "Superclass: " + superCl.getName() + "\n";

+ toInspect.add(superCl);

}

+

Class[] interfaceCl = classObj.getInterfaces();

+ if (interfaceCl.length > 0) {

+ str += indent(ind + 1) + "Interfaces: \n";

+ }

for (int i = 0; i < interfaceCl.length; i++) {

- str += "Interface: " + interfaceCl[i].getName() + "\n";

+ str += indent(ind + 2) + interfaceCl[i].getName() + "\n";

+ toInspect.add(interfaceCl[i]);

+ }

+

+ Method[] methods = classObj.getDeclaredMethods();

+ if (methods.length > 0) {

+ str += indent(ind + 1) + "Methods: \n";

+ }

+ for (int i = 0; i < methods.length; i++) {

+ str += indent(ind + 2) + "Method: " + methods[i].getName() + "\n";

+

+ Class[] exceps = methods[i].getExceptionTypes();

+ str += indent(ind + 3) + "Exceptions:\n";

+ if (exceps.length == 0) {

+ str += indent(ind + 4) + "None\n";

+ }

+ else {

+ for (int j = 0; j < exceps.length; j++) {

+ str += indent(ind + 4) + exceps[j].getName() + "\n";

+ }

+ }

+

+ Class[] params = methods[i].getParameterTypes();

+ str += indent(ind + 3) + "Parameters:\n";

+ if (params.length == 0) {

+ str += indent(ind + 4) + "None\n";

+ }

+ else {

+ for (int j = 0; j < params.length; j++) {

+ str += indent(ind + 4) + params[j].getTypeName() + ": " + params[j].getName() + "\n";

+ }

+ }

+

+ Class metRet = methods[i].getReturnType();

+ str += indent(ind + 3) + "Return Type: " + metRet.getTypeName() + "\n";

+

+ String[] mods = (Modifier.toString(methods[i].getModifiers())).split(" ");

+ str += indent(ind + 3) + "Modifiers:\n";

+ if (mods.length == 0) {

+ str += indent(ind + 4) + "None\n";

+ }

+ else {

+ for (int j = 0; j < mods.length; j++) {

+ str += indent(ind + 4) + mods[j] + "\n";

+ }

+ }

}

+ Constructor[] constr = classObj.getConstructors();

+ for (int i = 0; i < constr.length; i++) {

+ str += indent(ind + 1) + "Constructor: " + constr[i].getName() + "\n";

+

+ Class[] params = constr[i].getParameterTypes();

+ str += indent(ind + 2) + "Parameters:\n";

+ if (params.length == 0) {

+ str += indent(ind + 3) + "None\n";

+ }

+ else {

+ for (int j = 0; j < params.length; j++) {

+ str += indent(ind + 3) + params[j].getTypeName() + ": " + params[j].getName() + "\n";

+ }

+ }

+

+ String[] mods = (Modifier.toString(constr[i].getModifiers())).split(" ");

+ str += indent(ind + 2) + "Modifiers:\n";

+ if (mods.length == 0) {

+ str += indent(ind + 3) + "None\n";

+ }

+ else {

+ for (int j = 0; j < mods.length; j++) {

+ str += indent(ind + 3) + mods[j] + "\n";

+ }

+ }

+ }

+

+

+ Field[] fields = classObj.getDeclaredFields();

+ for (int i = 0; i < fields.length; i++) {

+ str += indent(ind + 1) + "Field: " + fields[i].getName() + "\n";

+

+ Class type = fields[i].getType();

+ str += indent(ind + 2) + "Type: " + type.getTypeName() + "\n";

+

+ String[] mods = (Modifier.toString(fields[i].getModifiers())).split(" ");

+ str += indent(ind + 2) + "Modifiers:\n";

+ if (mods.length == 0) {

+ str += indent(ind + 3) + "None\n";

+ }

+ else {

+ for (int j = 0; j < mods.length; j++) {

+ str += indent(ind + 3) + mods[j] + "\n";

+ }

+ }

+

+ //TODO make this work with arrays

+

+ if (!recursive) {

+ str += indent(ind + 2) + "Current value: " + fields[i].toString();

+ }

+ else {

+ //TODO finish the recursive case

+ }

+ }

@@ -32,4 +141,13 @@

System.out.println(str);

}

+

+

+ public String indent(int numIndents) {

+ String str = "";

+ for (int i = 0; i < numIndents; i++) {

+ str += " ";

+ }

+ return str;

+ }

}

commit c1ec9cce6313bb3c7173905e149102d326a0ef0c

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-02 21:35:58

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-02 21:35:58

Parent: 1a911224b6d574a041705e48ad6284d0740bbe6f (inspects methods and constructors. Fields half complete.)

Child: d0b080e2223786c964cd8dbdb14e3f70d03e19aa (Added array functionality during field inspection)

Branches: master, origin/master

Added recursive inspection for fields, superclass and interfaces

diff --git a/src/Inspector.java b/src/Inspector.java

index a7f98a7..79925ed 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -1,19 +1,32 @@

import java.lang.reflect.\*;

+import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

public class Inspector {

- List<Class> toInspect = new LinkedList<Class>();

List<Class> alreadyInspected = new LinkedList<Class>();

public void inspect(Object obj, boolean recursive) {

- inspectWorker(obj, recursive, 0);

+ String str;

+ if (obj == null) {

+ str = "Object provided is NULL\n";

+ }

+ else {

+ Class classObj = obj.getClass();

+ str = inspectWorker(classObj, obj, recursive, 0);

+ }

+ System.out.println(str);

}

- public void inspectWorker(Object obj, boolean recursive, int ind) {

+ public String inspectWorker(Class classObj, Object obj, boolean recursive, int ind) {

+ if (obj == null) {

+ return "NULL\n";

+ }

+

+ List<Class> toInspect = new ArrayList<Class>();

String str = "";

- Class classObj = obj.getClass();

+

alreadyInspected.add(classObj);

str += indent(ind) + "Declaring Class: " + classObj.getName() + "\n";

@@ -38,6 +51,7 @@

str += indent(ind + 1) + "Methods: \n";

}

for (int i = 0; i < methods.length; i++) {

+ methods[i].setAccessible(true);

str += indent(ind + 2) + "Method: " + methods[i].getName() + "\n";

Class[] exceps = methods[i].getExceptionTypes();

@@ -78,8 +92,9 @@

}

- Constructor[] constr = classObj.getConstructors();

+ Constructor[] constr = classObj.getDeclaredConstructors();

for (int i = 0; i < constr.length; i++) {

+ constr[i].setAccessible(true);

str += indent(ind + 1) + "Constructor: " + constr[i].getName() + "\n";

Class[] params = constr[i].getParameterTypes();

@@ -108,6 +123,7 @@

Field[] fields = classObj.getDeclaredFields();

for (int i = 0; i < fields.length; i++) {

+ fields[i].setAccessible(true);

str += indent(ind + 1) + "Field: " + fields[i].getName() + "\n";

Class type = fields[i].getType();

@@ -124,29 +140,85 @@

}

}

- //TODO make this work with arrays

-

- if (!recursive) {

- str += indent(ind + 2) + "Current value: " + fields[i].toString();

+ try {

+ if (fields[i].get(obj).getClass().isArray()) {

+ //TODO make this work with arrays

+

+ }

+ else {

+ str += indent(ind + 2) + "Current value: " + fields[i].get(obj).toString();

+ if (recursive && !(fields[i].get(obj).getClass().isPrimitive())) {

+ str += inspectWorker(fields[i].get(obj).getClass(), fields[i].get(obj), recursive, ind + 3);

+ }

+ }

+ } catch (IllegalArgumentException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ } catch (IllegalAccessException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

}

- else {

- //TODO finish the recursive case

- }

+ }

+

+ for (int i = 0; i < toInspect.size(); i++) {

+ //Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

+ str += "\n" + inspectWorker(toInspect.get(i), obj, recursive, ind);

}

-

-

- System.out.println(str);

+ return str;

}

+ /\*

+ public Object createSuperObjFromClass(Class cl, Object obj) {

+ Object supObj = null;

+ try {

+ Constructor c = cl.getConstructor(null);

+ c.setAccessible(true);

+ supObj = c.newInstance(null);

+ } catch (NoSuchMethodException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ } catch (SecurityException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ } catch (InstantiationException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ } catch (IllegalAccessException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ } catch (IllegalArgumentException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ } catch (InvocationTargetException e) {

+ // TODO Auto-generated catch block

+ e.printStackTrace();

+ }

+

+ Field[] fields = cl.getDeclaredFields();

+ for (int i = 0; i < fields.length; i++) {

+ fields[i].setAccessible(true);

+ try {

+ fields[i].set(supObj, fields[i].get(obj));

+ } catch (IllegalArgumentException e) {

+ //do nothing

+ } catch (IllegalAccessException e) {

+ // should not happen

+ e.printStackTrace();

+ }

+ }

+

+ return supObj;

+ }

+ //\*/

public String indent(int numIndents) {

String str = "";

for (int i = 0; i < numIndents; i++) {

- str += " ";

+ str += " ";

}

return str;

}

commit d0b080e2223786c964cd8dbdb14e3f70d03e19aa

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-02 21:58:18

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-02 21:58:18

Parent: c1ec9cce6313bb3c7173905e149102d326a0ef0c (Added recursive inspection for fields, superclass and interfaces)

Child: e17101f3d63c4ff07326d08cf1d4b0e5b7b4c164 (Add files via upload)

Branches: master, origin/master

Added array functionality during field inspection

diff --git a/src/Inspector.java b/src/Inspector.java

index 79925ed..fcc00d6 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -143,7 +143,16 @@

try {

if (fields[i].get(obj).getClass().isArray()) {

//TODO make this work with arrays

-

+ Object arr = fields[i].get(obj);

+ int len = Array.getLength(arr);

+ str += indent(ind + 2) + "Array length: " + len + "\n";

+ str += indent(ind + 2) + "Current values: \n";

+ for (int k = 0; k < len; k++) {

+ str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

+ if (recursive && !(Array.get(arr, k).getClass().isPrimitive())) {

+ str += inspectWorker(Array.get(arr, k).getClass(), arr, recursive, ind + 4);

+ }

+ }

}

else {

str += indent(ind + 2) + "Current value: " + fields[i].get(obj).toString();

commit c8d2506feba39ca902cd56291b675a462b9e5754

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 09:39:10

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 09:39:10

Parent: e17101f3d63c4ff07326d08cf1d4b0e5b7b4c164 (Add files via upload)

Child: 268df2feeda325e479fafcaae42b9ecb20bde924 (fixed bugs in inspectWorker. superclass/interface correctly inspected)

Branches: master, origin/master

Fixed bugs in inspectWorker. Added createSuperObjFromClass

diff --git a/src/Inspector.java b/src/Inspector.java

index fcc00d6..0e97859 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -5,7 +5,7 @@

public class Inspector {

- List<Class> alreadyInspected = new LinkedList<Class>();

+ List<Class<?>> alreadyInspected = new LinkedList<Class<?>>();

public void inspect(Object obj, boolean recursive) {

String str;

@@ -13,31 +13,31 @@

str = "Object provided is NULL\n";

}

else {

- Class classObj = obj.getClass();

+ Class<?> classObj = obj.getClass();

str = inspectWorker(classObj, obj, recursive, 0);

}

System.out.println(str);

}

- public String inspectWorker(Class classObj, Object obj, boolean recursive, int ind) {

- if (obj == null) {

+ public String inspectWorker(Class<?> classObj, Object obj, boolean recursive, int ind) {

+ /\*if (obj == null) {

return "NULL\n";

- }

+ }//\*/

- List<Class> toInspect = new ArrayList<Class>();

+ List<Class<?>> toInspect = new ArrayList<Class<?>>();

String str = "";

alreadyInspected.add(classObj);

- str += indent(ind) + "Declaring Class: " + classObj.getName() + "\n";

+ str += indent(ind) + "Declared Class: " + classObj.getName() + "\n";

- Class superCl = classObj.getSuperclass();

+ Class<?> superCl = classObj.getSuperclass();

if (superCl != null) {

str += indent(ind + 1) + "Superclass: " + superCl.getName() + "\n";

toInspect.add(superCl);

}

- Class[] interfaceCl = classObj.getInterfaces();

+ Class<?>[] interfaceCl = classObj.getInterfaces();

if (interfaceCl.length > 0) {

str += indent(ind + 1) + "Interfaces: \n";

}

@@ -54,29 +54,31 @@

methods[i].setAccessible(true);

str += indent(ind + 2) + "Method: " + methods[i].getName() + "\n";

- Class[] exceps = methods[i].getExceptionTypes();

- str += indent(ind + 3) + "Exceptions:\n";

+ Class<?>[] exceps = methods[i].getExceptionTypes();

+ str += indent(ind + 3) + "Exceptions: ";

if (exceps.length == 0) {

- str += indent(ind + 4) + "None\n";

+ str += "None\n";

}

else {

for (int j = 0; j < exceps.length; j++) {

- str += indent(ind + 4) + exceps[j].getName() + "\n";

+ str += exceps[j].getName() + " ";

}

+ str += "\n";

}

- Class[] params = methods[i].getParameterTypes();

- str += indent(ind + 3) + "Parameters:\n";

+ Class<?>[] params = methods[i].getParameterTypes();

+ str += indent(ind + 3) + "Parameters: ";

if (params.length == 0) {

- str += indent(ind + 4) + "None\n";

+ str += "None\n";

}

else {

for (int j = 0; j < params.length; j++) {

- str += indent(ind + 4) + params[j].getTypeName() + ": " + params[j].getName() + "\n";

+ str += params[j].getTypeName() + " ";

}

+ str += "\n";

}

- Class metRet = methods[i].getReturnType();

+ Class<?> metRet = methods[i].getReturnType();

str += indent(ind + 3) + "Return Type: " + metRet.getTypeName() + "\n";

String[] mods = (Modifier.toString(methods[i].getModifiers())).split(" ");

@@ -85,27 +87,32 @@

str += indent(ind + 4) + "None\n";

}

else {

+ str += indent(ind + 4);

for (int j = 0; j < mods.length; j++) {

- str += indent(ind + 4) + mods[j] + "\n";

+ str += mods[j] + " ";

}

+ str += "\n";

}

}

- Constructor[] constr = classObj.getDeclaredConstructors();

+ Constructor<?>[] constr = classObj.getDeclaredConstructors();

for (int i = 0; i < constr.length; i++) {

- constr[i].setAccessible(true);

+ if (Modifier.isPrivate(constr[i].getModifiers())) {

+ //constr[i].setAccessible(true);

+ }

str += indent(ind + 1) + "Constructor: " + constr[i].getName() + "\n";

- Class[] params = constr[i].getParameterTypes();

- str += indent(ind + 2) + "Parameters:\n";

+ Class<?>[] params = constr[i].getParameterTypes();

+ str += indent(ind + 2) + "Parameters: ";

if (params.length == 0) {

- str += indent(ind + 3) + "None\n";

+ str += "None\n";

}

else {

for (int j = 0; j < params.length; j++) {

- str += indent(ind + 3) + params[j].getTypeName() + ": " + params[j].getName() + "\n";

+ str += params[j].getTypeName() + " ";

}

+ str += "\n";

}

String[] mods = (Modifier.toString(constr[i].getModifiers())).split(" ");

@@ -114,9 +121,13 @@

str += indent(ind + 3) + "None\n";

}

else {

- for (int j = 0; j < mods.length; j++) {

- str += indent(ind + 3) + mods[j] + "\n";

+ if (mods.length >= 0) {

+ str += indent(ind + 3) + mods[0];

}

+ for (int j = 1; j < mods.length; j++) {

+ str += ", " + mods[j];

+ }

+ str += "\n";

}

}

@@ -126,96 +137,142 @@

fields[i].setAccessible(true);

str += indent(ind + 1) + "Field: " + fields[i].getName() + "\n";

- Class type = fields[i].getType();

+ Class<?> type = fields[i].getType();

str += indent(ind + 2) + "Type: " + type.getTypeName() + "\n";

String[] mods = (Modifier.toString(fields[i].getModifiers())).split(" ");

- str += indent(ind + 2) + "Modifiers:\n";

+ str += indent(ind + 2) + "Modifiers: ";

if (mods.length == 0) {

- str += indent(ind + 3) + "None\n";

+ str += "None\n";

}

else {

for (int j = 0; j < mods.length; j++) {

- str += indent(ind + 3) + mods[j] + "\n";

+ str += mods[j] + " ";

}

+ str += "\n";

}

try {

- if (fields[i].get(obj).getClass().isArray()) {

- //TODO make this work with arrays

+ if (fields[i].get(obj) == null) {

+ str += indent(ind + 2) + "Current value: null\n";

+ }

+ else if (fields[i].get(obj).getClass().isArray()) {

Object arr = fields[i].get(obj);

int len = Array.getLength(arr);

str += indent(ind + 2) + "Array length: " + len + "\n";

str += indent(ind + 2) + "Current values: \n";

for (int k = 0; k < len; k++) {

str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

- if (recursive && !(Array.get(arr, k).getClass().isPrimitive())) {

- str += inspectWorker(Array.get(arr, k).getClass(), arr, recursive, ind + 4);

- }

+ if (recursive && !(Array.get(arr, k).getClass().isPrimitive())

+ && hasNotInspected(Array.get(arr, k).getClass())) {

+ str += inspectWorker(Array.get(arr, k).getClass(), Array.get(arr, k), recursive, ind + 4);

+ }//\*/

}

}

else {

- str += indent(ind + 2) + "Current value: " + fields[i].get(obj).toString();

- if (recursive && !(fields[i].get(obj).getClass().isPrimitive())) {

+ str += indent(ind + 2) + "Current value: " + fields[i].get(obj).toString() + "\n";

+ /\*if (recursive && !(fields[i].get(obj).getClass().isPrimitive())

+ && hasNotInspected(fields[i].get(obj).getClass())) {

str += inspectWorker(fields[i].get(obj).getClass(), fields[i].get(obj), recursive, ind + 3);

- }

+ }//\*/

}

} catch (IllegalArgumentException e) {

- // TODO Auto-generated catch block

+ System.out.println("IllegalArgumentException classObj: " + classObj.toString()

+ + "field: " + fields[i].toString()

+ + ", obj: " + obj.toString() + "\n");

e.printStackTrace();

+ System.exit(-1);

} catch (IllegalAccessException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

+ //System.exit(-1);

}

}

+ str += indent(ind) + "========================================\n";

+ ///\*

for (int i = 0; i < toInspect.size(); i++) {

- //Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

- str += "\n" + inspectWorker(toInspect.get(i), obj, recursive, ind);

+ Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

+ if (supObj != null) {

+ supObj = obj;

+ }

+ else if (hasNotInspected(toInspect.get(i))) {

+ str += "\n" + inspectWorker(toInspect.get(i),

+ supObj,

+ recursive,

+ ind);

+ }

}

-

-

-

-

+ //\*/

return str;

}

- /\*

- public Object createSuperObjFromClass(Class cl, Object obj) {

+ public boolean hasNotInspected(Class<?> c) {

+ boolean ret = true;

+

+ for (int i = 0; i < alreadyInspected.size() && ret; i++) {

+ if (alreadyInspected.get(i).equals(c)) {

+ ret = false;

+ }

+ }

+

+ return ret;

+ }

+

+ ///\*

+ public Object createSuperObjFromClass(Class<?> cl, Object obj) {

Object supObj = null;

+ if (cl.isInterface()) {

+ return supObj;

+ }

try {

- Constructor c = cl.getConstructor(null);

+ Constructor<?> c = cl.getDeclaredConstructor(new Class<?>[] {});

c.setAccessible(true);

- supObj = c.newInstance(null);

+ supObj = c.newInstance(new Object[] {});

} catch (NoSuchMethodException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

+ //System.exit(-1);//\*/

} catch (SecurityException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

+ //System.exit(-1);

} catch (InstantiationException e) {

- // TODO Auto-generated catch block

+ /\*System.out.println(cl.getName());

+ System.out.println("interface?: " + cl.isInterface());

e.printStackTrace();

+ System.exit(-1);\*/

+ // if could not be create instance, return null.

} catch (IllegalAccessException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

+ //System.exit(-1);

} catch (IllegalArgumentException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

+ //System.exit(-1);

} catch (InvocationTargetException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

+ //System.exit(-1);//\*/

}

Field[] fields = cl.getDeclaredFields();

for (int i = 0; i < fields.length; i++) {

fields[i].setAccessible(true);

try {

- fields[i].set(supObj, fields[i].get(obj));

+ Object temp = obj.getClass().getDeclaredField(fields[i].getName()).get(obj);

+ fields[i].set(supObj, temp);

} catch (IllegalArgumentException e) {

//do nothing

} catch (IllegalAccessException e) {

// should not happen

+ e.printStackTrace();

+ } catch (NoSuchFieldException e) {

+ // do nothing

+ //e.printStackTrace();

+ } catch (SecurityException e) {

+ // TODO Auto-generated catch block

e.printStackTrace();

}

}

@@ -227,7 +284,7 @@

public String indent(int numIndents) {

String str = "";

for (int i = 0; i < numIndents; i++) {

- str += " ";

+ str += " | ";

}

return str;

}

commit 268df2feeda325e479fafcaae42b9ecb20bde924

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 13:19:07

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 13:19:07

Parent: c8d2506feba39ca902cd56291b675a462b9e5754 (Fixed bugs in inspectWorker. Added createSuperObjFromClass)

Child: 94bb9f4a5661776c7147393a3018e65ca39633fd (Applied Extract Method to inspectArray prevent code duplication)

Branches: master, origin/master

fixed bugs in inspectWorker. superclass/interface correctly inspected

null and primitives also dealt with.

diff --git a/src/Inspector.java b/src/Inspector.java

index 0e97859..9f6e4d4 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -46,6 +46,10 @@

toInspect.add(interfaceCl[i]);

}

+ /\*if (classObj.isArray() && hasNotInspected(classObj.getComponentType())) {

+ str += inspectArray(obj);

+ }\*/

+

Method[] methods = classObj.getDeclaredMethods();

if (methods.length > 0) {

str += indent(ind + 1) + "Methods: \n";

@@ -153,26 +157,37 @@

}

try {

- if (fields[i].get(obj) == null) {

+ if (obj == null || fields[i] == null || fields[i].get(obj) == null) {

str += indent(ind + 2) + "Current value: null\n";

- }

- else if (fields[i].get(obj).getClass().isArray()) {

+ }//\*/

+ else if (fields[i].getType().isArray()) {

Object arr = fields[i].get(obj);

int len = Array.getLength(arr);

str += indent(ind + 2) + "Array length: " + len + "\n";

str += indent(ind + 2) + "Current values: \n";

for (int k = 0; k < len; k++) {

- str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

- if (recursive && !(Array.get(arr, k).getClass().isPrimitive())

+ if (Array.get(arr, k) == null)

+ str += indent(ind + 3) + "null \n";

+ else {

+ str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

+ }

+ if (Array.get(arr, k) != null && hasNotInspected(fields[i].getType().getComponentType())) {

+ str += inspectWorker(fields[i].getType().getComponentType(), null, recursive, ind + 4);

+ }

+ else if (recursive && (Array.get(arr, k) != null)

+ && !(isPrimOrWrapper(Array.get(arr, k).getClass()))

&& hasNotInspected(Array.get(arr, k).getClass())) {

str += inspectWorker(Array.get(arr, k).getClass(), Array.get(arr, k), recursive, ind + 4);

}//\*/

}

}

else {

+ if (fields[i].get(obj) == null) {

+ str += indent(ind + 2) + "Current value: null\n";

+ }

str += indent(ind + 2) + "Current value: " + fields[i].get(obj).toString() + "\n";

- /\*if (recursive && !(fields[i].get(obj).getClass().isPrimitive())

- && hasNotInspected(fields[i].get(obj).getClass())) {

+ if (recursive && !(isPrimOrWrapper(fields[i].getType()))

+ && hasNotInspected(fields[i].getType())) {

str += inspectWorker(fields[i].get(obj).getClass(), fields[i].get(obj), recursive, ind + 3);

}//\*/

}

@@ -190,10 +205,11 @@

}

str += indent(ind) + "========================================\n";

+

///\*

for (int i = 0; i < toInspect.size(); i++) {

Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

- if (supObj != null) {

+ if (supObj == null) {

supObj = obj;

}

else if (hasNotInspected(toInspect.get(i))) {

@@ -222,7 +238,7 @@

///\*

public Object createSuperObjFromClass(Class<?> cl, Object obj) {

Object supObj = null;

- if (cl.isInterface()) {

+ if (cl.isInterface() || obj == null) {

return supObj;

}

try {

@@ -266,14 +282,13 @@

} catch (IllegalArgumentException e) {

//do nothing

} catch (IllegalAccessException e) {

- // should not happen

- e.printStackTrace();

+ //e.printStackTrace();

} catch (NoSuchFieldException e) {

// do nothing

//e.printStackTrace();

} catch (SecurityException e) {

// TODO Auto-generated catch block

- e.printStackTrace();

+ //e.printStackTrace();

}

}

@@ -288,4 +303,11 @@

}

return str;

}

+

+ public boolean isPrimOrWrapper(Class<?> c) {

+ return ((c.isPrimitive() && c != void.class) ||

+ c == Double.class || c == Float.class || c == Long.class ||

+ c == Integer.class || c == Short.class || c == Character.class ||

+ c == Byte.class || c == Boolean.class);

+ }

}

commit 94bb9f4a5661776c7147393a3018e65ca39633fd

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:39:49

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:39:49

Parent: 268df2feeda325e479fafcaae42b9ecb20bde924 (fixed bugs in inspectWorker. superclass/interface correctly inspected)

Child: 584017b9b046eaaf87fb6a204e549c8488ef4149 (Applied Extract Method to inspectFields )

Branches: master, origin/master

Applied Extract Method to inspectArray prevent code duplication

diff --git a/src/Inspector.java b/src/Inspector.java

index 9f6e4d4..c0a14da 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -46,9 +46,9 @@

toInspect.add(interfaceCl[i]);

}

- /\*if (classObj.isArray() && hasNotInspected(classObj.getComponentType())) {

- str += inspectArray(obj);

- }\*/

+ if (classObj.isArray() && hasNotInspected(classObj.getComponentType())) {

+ str += inspectArray(recursive, ind, classObj, obj);

+ }//\*/

Method[] methods = classObj.getDeclaredMethods();

if (methods.length > 0) {

@@ -162,24 +162,7 @@

}//\*/

else if (fields[i].getType().isArray()) {

Object arr = fields[i].get(obj);

- int len = Array.getLength(arr);

- str += indent(ind + 2) + "Array length: " + len + "\n";

- str += indent(ind + 2) + "Current values: \n";

- for (int k = 0; k < len; k++) {

- if (Array.get(arr, k) == null)

- str += indent(ind + 3) + "null \n";

- else {

- str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

- }

- if (Array.get(arr, k) != null && hasNotInspected(fields[i].getType().getComponentType())) {

- str += inspectWorker(fields[i].getType().getComponentType(), null, recursive, ind + 4);

- }

- else if (recursive && (Array.get(arr, k) != null)

- && !(isPrimOrWrapper(Array.get(arr, k).getClass()))

- && hasNotInspected(Array.get(arr, k).getClass())) {

- str += inspectWorker(Array.get(arr, k).getClass(), Array.get(arr, k), recursive, ind + 4);

- }//\*/

- }

+ str += inspectArray(recursive, ind, fields[i].getType().getComponentType(), arr);

}

else {

if (fields[i].get(obj) == null) {

@@ -222,6 +205,50 @@

//\*/

return str;

}

+

+ public String inspectArray(boolean recursive, int ind, Class<?> arrayClass, Object arr) {

+ String str = "";

+ Class<?> arrayType = arrayClass.getComponentType();

+ if (arr != null) {

+ str += indent(ind + 2) + "Array dimensions: " + getDim(arr) + "\n";

+ }

+ int len = 0;

+ if (arr != null) {

+ len = Array.getLength(arr);

+ str += indent(ind + 2) + "Array length (1st dimension): " + len + "\n";

+ }

+ else {

+ str += indent(ind + 2) + "Array length: could not be determined (array is null)\n";

+ }//\*/

+ str += indent(ind + 2) + "Current values: \n";

+ for (int k = 0; k < len; k++) {

+ if (Array.get(arr, k) == null)

+ str += indent(ind + 3) + "null \n";

+ else {

+ str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

+ }

+ if (Array.get(arr, k) != null && hasNotInspected(arrayType)) {

+ str += inspectWorker(arrayType, null, recursive, ind + 4);

+ }

+ else if (recursive && (Array.get(arr, k) != null)

+ && !(isPrimOrWrapper(Array.get(arr, k).getClass()))

+ && hasNotInspected(Array.get(arr, k).getClass())) {

+ str += inspectWorker(Array.get(arr, k).getClass(), Array.get(arr, k), recursive, ind + 4);

+ }//\*/

+ }

+ return str;

+ }

+

+ //public int getDim(Class<?> arrayClass) {

+ public int getDim(Object array) {

+ int dim = 0;

+ Class<?> cl = array.getClass();

+ while (cl.isArray()) {

+ dim++;

+ cl = cl.getComponentType();

+ }

+ return dim;

+ }

public boolean hasNotInspected(Class<?> c) {

boolean ret = true;

commit 584017b9b046eaaf87fb6a204e549c8488ef4149

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:44:24

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:44:24

Parent: 94bb9f4a5661776c7147393a3018e65ca39633fd (Applied Extract Method to inspectArray prevent code duplication)

Child: 70d5bb98e2854542b5851db45b066ddfbf7b0f9a (Applied extract method to inspectConstructor )

Branches: master, origin/master

Applied Extract Method to inspectFields

done to breakup inspectWorker (long Method)

diff --git a/src/Inspector.java b/src/Inspector.java

index c0a14da..125297a 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -136,6 +136,28 @@

}

diff --git a/src/Inspector.java b/src/Inspector.java

index c0a14da..125297a 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -136,6 +136,28 @@

}

+ str = inspectFields(classObj, obj, recursive, ind, str);

+

+ str += indent(ind) + "========================================\n";

+

+ ///\*

+ for (int i = 0; i < toInspect.size(); i++) {

+ Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

+ if (supObj == null) {

+ supObj = obj;

+ }

+ else if (hasNotInspected(toInspect.get(i))) {

+ str += "\n" + inspectWorker(toInspect.get(i),

+ supObj,

+ recursive,

+ ind);

+ }

+ }

+ //\*/

+ return str;

+ }

+

+ public String inspectFields(Class<?> classObj, Object obj, boolean recursive, int ind, String str) {

Field[] fields = classObj.getDeclaredFields();

for (int i = 0; i < fields.length; i++) {

fields[i].setAccessible(true);

@@ -186,23 +208,6 @@

//System.exit(-1);

}

}

-

- str += indent(ind) + "========================================\n";

-

- ///\*

- for (int i = 0; i < toInspect.size(); i++) {

- Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

- if (supObj == null) {

- supObj = obj;

- }

- else if (hasNotInspected(toInspect.get(i))) {

- str += "\n" + inspectWorker(toInspect.get(i),

- supObj,

- recursive,

- ind);

- }

- }

- //\*/

return str;

}

commit 70d5bb98e2854542b5851db45b066ddfbf7b0f9a

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:45:54

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:45:54

Parent: 584017b9b046eaaf87fb6a204e549c8488ef4149 (Applied Extract Method to inspectFields )

Child: f70a27bdf46c24b56754f5d360078c707547bf1d (Applied extract method to inspectMethods)

Branches: master, origin/master

Applied extract method to inspectConstructor

to breakup inspectWorker (long method)

diff --git a/src/Inspector.java b/src/Inspector.java

index 125297a..7cce711 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -100,6 +100,31 @@

}

+ str = inspectConstructors(classObj, ind, str);

+

+

+ str = inspectFields(classObj, obj, recursive, ind, str);

+

+ str += indent(ind) + "========================================\n";

+

+ ///\*

+ for (int i = 0; i < toInspect.size(); i++) {

+ Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

+ if (supObj == null) {

+ supObj = obj;

+ }

+ else if (hasNotInspected(toInspect.get(i))) {

+ str += "\n" + inspectWorker(toInspect.get(i),

+ supObj,

+ recursive,

+ ind);

+ }

+ }

+ //\*/

+ return str;

+ }

+

+ public String inspectConstructors(Class<?> classObj, int ind, String str) {

Constructor<?>[] constr = classObj.getDeclaredConstructors();

for (int i = 0; i < constr.length; i++) {

if (Modifier.isPrivate(constr[i].getModifiers())) {

@@ -134,26 +159,6 @@

str += "\n";

}

}

-

-

- str = inspectFields(classObj, obj, recursive, ind, str);

-

- str += indent(ind) + "========================================\n";

-

- ///\*

- for (int i = 0; i < toInspect.size(); i++) {

- Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

- if (supObj == null) {

- supObj = obj;

- }

- else if (hasNotInspected(toInspect.get(i))) {

- str += "\n" + inspectWorker(toInspect.get(i),

- supObj,

- recursive,

- ind);

- }

- }

- //\*/

return str;

}

commit f70a27bdf46c24b56754f5d360078c707547bf1d

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:47:16

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 19:47:16

Parent: 70d5bb98e2854542b5851db45b066ddfbf7b0f9a (Applied extract method to inspectConstructor )

Child: a717e39d306a2eb8b75ebcf93b58d0b6b5bc7a26 (Repaired interface inspection through recursive descent.)

Branches: master, origin/master

Applied extract method to inspectMethods

inspectWorker was too long and had obvious sections

diff --git a/src/Inspector.java b/src/Inspector.java

index 7cce711..f00dfba 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -50,6 +50,34 @@

str += inspectArray(recursive, ind, classObj, obj);

}//\*/

+ str = inspectMethods(classObj, ind, str);

+

+

+ str = inspectConstructors(classObj, ind, str);

+

+

+ str = inspectFields(classObj, obj, recursive, ind, str);

+

+ str += indent(ind) + "========================================\n";

+

+ ///\*

+ for (int i = 0; i < toInspect.size(); i++) {

+ Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

+ if (supObj == null) {

+ supObj = obj;

+ }

+ else if (hasNotInspected(toInspect.get(i))) {

+ str += "\n" + inspectWorker(toInspect.get(i),

+ supObj,

+ recursive,

+ ind);

+ }

+ }

+ //\*/

+ return str;

+ }

+

+ public String inspectMethods(Class<?> classObj, int ind, String str) {

Method[] methods = classObj.getDeclaredMethods();

if (methods.length > 0) {

str += indent(ind + 1) + "Methods: \n";

@@ -98,29 +126,6 @@

str += "\n";

}

}

-

-

- str = inspectConstructors(classObj, ind, str);

-

-

- str = inspectFields(classObj, obj, recursive, ind, str);

-

- str += indent(ind) + "========================================\n";

-

- ///\*

- for (int i = 0; i < toInspect.size(); i++) {

- Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

- if (supObj == null) {

- supObj = obj;

- }

- else if (hasNotInspected(toInspect.get(i))) {

- str += "\n" + inspectWorker(toInspect.get(i),

- supObj,

- recursive,

- ind);

- }

- }

- //\*/

return str;

}

commit a717e39d306a2eb8b75ebcf93b58d0b6b5bc7a26

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 20:17:57

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 20:17:57

Parent: f70a27bdf46c24b56754f5d360078c707547bf1d (Applied extract method to inspectMethods)

Child: c2de0b9ab7e31b0d850f4d733cb05b9c9b8929cb (Tests to evaluate Inspector. Reflection order is inconsistent!!!)

Branches: master, origin/master

Repaired interface inspection through recursive descent.

diff --git a/src/Inspector.java b/src/Inspector.java

index f00dfba..5d399f5 100644

--- a/src/Inspector.java

+++ b/src/Inspector.java

@@ -63,10 +63,7 @@

///\*

for (int i = 0; i < toInspect.size(); i++) {

Object supObj = createSuperObjFromClass(toInspect.get(i), obj);

- if (supObj == null) {

- supObj = obj;

- }

- else if (hasNotInspected(toInspect.get(i))) {

+ if (hasNotInspected(toInspect.get(i))) {

str += "\n" + inspectWorker(toInspect.get(i),

supObj,

recursive,

@@ -194,7 +191,7 @@

}//\*/

else if (fields[i].getType().isArray()) {

Object arr = fields[i].get(obj);

- str += inspectArray(recursive, ind, fields[i].getType().getComponentType(), arr);

+ str += inspectArray(recursive, ind, fields[i].getType(), arr);

}

else {

if (fields[i].get(obj) == null) {

@@ -243,7 +240,7 @@

str += indent(ind + 3) + Array.get(arr, k).toString() + "\n";

}

if (Array.get(arr, k) != null && hasNotInspected(arrayType)) {

- str += inspectWorker(arrayType, null, recursive, ind + 4);

+ str += inspectWorker(arrayType, Array.get(arr, k), recursive, ind + 4);

}

else if (recursive && (Array.get(arr, k) != null)

&& !(isPrimOrWrapper(Array.get(arr, k).getClass()))

@@ -281,7 +278,7 @@

public Object createSuperObjFromClass(Class<?> cl, Object obj) {

Object supObj = null;

if (cl.isInterface() || obj == null) {

- return supObj;

+ return obj;

}

try {

Constructor<?> c = cl.getDeclaredConstructor(new Class<?>[] {});

commit c2de0b9ab7e31b0d850f4d733cb05b9c9b8929cb

Author: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 22:46:41

Committer: ttdjohns <32712899+ttdjohns@users.noreply.github.com> 2019-03-07 22:46:41

Parent: a717e39d306a2eb8b75ebcf93b58d0b6b5bc7a26 (Repaired interface inspection through recursive descent.)

Child: 923d72779a0ff3d450b1d69a819f3b82b2e744f9 (script.txt of the output of the driver program)

Branches: master, origin/master

Tests to evaluate Inspector. Reflection order is inconsistent!!!

diff --git a/src/InspectorTests.java b/src/InspectorTests.java

new file mode 100644

index 0000000..591cba9

--- /dev/null

+++ b/src/InspectorTests.java

@@ -0,0 +1,102 @@

+import static org.junit.Assert.\*;

+

+import org.junit.Before;

+import org.junit.After;

+import org.junit.Test;

+

+/\*\*

+ \* Test cases for Inspector.java

+ \*

+ \* NOTE!!!!!! reflection is very inconsistent with the order that it gets fields/methods in.

+ \*

+ \* @author ttdjo

+ \*

+ \*/

+

+public class InspectorTests {

+ Object c1a;

+ Object c1b;

+ Object c2;

+ Inspector insp;

+

+ @Before

+ public void setup() {

+ c1a = new C1();

+ c1b = new C1(7);

+ c2 = new C2();

+ insp = new Inspector();

+ }

+

+ @After

+ public void teardown() {

+ c1a = null;

+ c1b = null;

+ c2 = null;

+ insp = null;

+

+ }

+

+ @Test

+ public void testInpMethods() {

+ assertEquals(insp.inspectMethods(c1a.getClass(), 0, ""), c1aM\_printout);

+ assertEquals(insp.inspectMethods(c1a.getClass(), 0, ""), c1bM\_printout);

+ assertEquals(insp.inspectMethods(c2.getClass(), 0, ""), c2M\_printout);

+ }

+

+ @Test

+ public void testInpConstructors() {

+ //System.out.println(insp.inspectConstructors(c2.getClass(), 0, ""));

+ assertEquals(insp.inspectConstructors(c1a.getClass(), 0, ""), c1aC\_printout);

+ assertEquals(insp.inspectConstructors(c1b.getClass(), 0, ""), c1bC\_printout);

+ assertEquals(insp.inspectConstructors(c2.getClass(), 0, ""), c2C\_printout);

+ }

+

+ @Test

+ public void testInpFields() {

+ //System.out.println(insp.inspectFields(c2.getClass(), c2, false, 0, ""));

+ assertEquals(insp.inspectFields(c1a.getClass(), c1a, false, 0, ""), c1aFnr\_printout);

+ assertEquals(insp.inspectFields(c1b.getClass(), c1b, false, 0, ""), c1bFnr\_printout);

+ assertEquals(insp.inspectFields(c2.getClass(), c2, false, 0, ""), c2Fnr\_printout\_alternitive);

+ //assertTrue(true);

+ }

+

+ @Test

+ public void testInpect() {

+ //System.out.println(insp.inspectWorker(c2.getClass(), c2, false, 0));

+ assertEquals(c1aInr\_printout, insp.inspectWorker(c1a.getClass(), c1a, false, 0));

+ assertEquals(c2Inr\_printout, insp.inspectWorker(c2.getClass(), c2, false, 0));

+ assertTrue(true);

+ }

+

+ @Test

+ public void testInpspectR() {

+ //System.out.println(insp.inspectMethods(c2.getClass(), 0, ""));

+ //assertEquals(insp.inspectMethods(c1a.getClass(), 0, ""), c1aInr\_printout);

+ //assertEquals(insp.inspectMethods(c1a.getClass(), 0, ""), c1bInr\_printout);

+ //assertEquals(insp.inspectMethods(c2.getClass(), 0, ""), c2Inr\_printout);

+ assertTrue(true);

+ }

+

+ String c1aM\_printout = " | Methods: \n | | Method: aThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public \n";

+ String c1bM\_printout = " | Methods: \n | | Method: aThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public \n";

+ String c2M\_printout = " | Methods: \n | | Method: aThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public \n | | Method: anotherThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.String\n | | | Modifiers:\n | | | | public \n";

+

+ String c1aC\_printout = " | Constructor: C1\n | | Parameters: None\n | | Modifiers:\n | | | public\n | Constructor: C1\n | | Parameters: int \n | | Modifiers:\n | | | public\n";

+ String c1bC\_printout = " | Constructor: C1\n | | Parameters: None\n | | Modifiers:\n | | | public\n | Constructor: C1\n | | Parameters: int \n | | Modifiers:\n | | | public\n";

+ String c2C\_printout = " | Constructor: C2\n | | Parameters: None\n | | Modifiers:\n | | | public\n";

+

+ String c1aFnr\_printout = " | Field: a\n | | Type: int\n | | Modifiers: public \n | | Current value: 0\n | Field: x\n | | Type: long\n | | Modifiers: private \n | | Current value: 3\n";

+ String c1bFnr\_printout = " | Field: a\n | | Type: int\n | | Modifiers: public \n | | Current value: 7\n | Field: x\n | | Type: long\n | | Modifiers: private \n | | Current value: 3\n";

+ String c2Fnr\_printout = " | Field: c\n | | Type: int[]\n | | Modifiers: public \n | | Array dimensions: 1\n | | Array length (1st dimension): 2\n | | Current values: \n | | | 0\n | | | 1\n | Field: d\n | | Type: double\n | | Modifiers: public final \n | | Current value: 1.1\n";

+ String c2Fnr\_printout\_alternitive = " | Field: c\n | | Type: int[]\n | | Modifiers: public \n | | Array dimensions: 1\n | | Array length (1st dimension): 2\n | | Current values: \n | | | 0\n | | | | Declared Class: int\n | | | | ========================================\n | | | 1\n | Field: d\n | | Type: double\n | | Modifiers: public final \n | | Current value: 1.1\n";

+

+ String c1aInr\_printout = "Declared Class: C1\n | Superclass: java.lang.Object\n | Methods: \n | | Method: aThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public \n | Constructor: C1\n | | Parameters: None\n | | Modifiers:\n | | | public\n | Constructor: C1\n | | Parameters: int \n | | Modifiers:\n | | | public\n | Field: a\n | | Type: int\n | | Modifiers: public \n | | Current value: 0\n | Field: x\n | | Type: long\n | | Modifiers: private \n | | Current value: 3\n========================================\n\nDeclared Class: java.lang.Object\n | Methods: \n | | Method: finalize\n | | | Exceptions: java.lang.Throwable \n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | protected \n | | Method: wait\n | | | Exceptions: java.lang.InterruptedException \n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public final \n | | Method: wait\n | | | Exceptions: java.lang.InterruptedException \n | | | Parameters: long int \n | | | Return Type: void\n | | | Modifiers:\n | | | | public final \n | | Method: wait\n | | | Exceptions: java.lang.InterruptedException \n | | | Parameters: long \n | | | Return Type: void\n | | | Modifiers:\n | | | | public final native \n | | Method: equals\n | | | Exceptions: None\n | | | Parameters: java.lang.Object \n | | | Return Type: boolean\n | | | Modifiers:\n | | | | public \n | | Method: toString\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.String\n | | | Modifiers:\n | | | | public \n | | Method: hashCode\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: int\n | | | Modifiers:\n | | | | public native \n | | Method: getClass\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.Class\n | | | Modifiers:\n | | | | public final native \n | | Method: clone\n | | | Exceptions: java.lang.CloneNotSupportedException \n | | | Parameters: None\n | | | Return Type: java.lang.Object\n | | | Modifiers:\n | | | | protected native \n | | Method: notify\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public final native \n | | Method: notifyAll\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public final native \n | | Method: registerNatives\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | private static native \n | Constructor: java.lang.Object\n | | Parameters: None\n | | Modifiers:\n | | | public\n========================================\n";

+ String c1bInr\_printout = "";

+ String c2Inr\_printout = "Declared Class: C2\n | Superclass: C1\n | Interfaces: \n | | MyI\n | Methods: \n | | Method: anotherThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.String\n | | | Modifiers:\n | | | | public \n | | Method: aThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public \n | Constructor: C2\n | | Parameters: None\n | | Modifiers:\n | | | public\n | Field: c\n | | Type: int[]\n | | Modifiers: public \n | | Array dimensions: 1\n | | Array length (1st dimension): 2\n | | Current values: \n | | | 0\n | | | | Declared Class: int\n | | | | ========================================\n | | | 1\n | Field: d\n | | Type: double\n | | Modifiers: public final \n | | Current value: 1.1\n========================================\n\nDeclared Class: C1\n | Superclass: java.lang.Object\n | Methods: \n | | Method: aThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public \n | Constructor: C1\n | | Parameters: None\n | | Modifiers:\n | | | public\n | Constructor: C1\n | | Parameters: int \n | | Modifiers:\n | | | public\n | Field: a\n | | Type: int\n | | Modifiers: public \n | | Current value: 0\n | Field: x\n | | Type: long\n | | Modifiers: private \n | | Current value: 3\n========================================\n\nDeclared Class: java.lang.Object\n | Methods: \n | | Method: finalize\n | | | Exceptions: java.lang.Throwable \n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | protected \n | | Method: wait\n | | | Exceptions: java.lang.InterruptedException \n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public final \n | | Method: wait\n | | | Exceptions: java.lang.InterruptedException \n | | | Parameters: long int \n | | | Return Type: void\n | | | Modifiers:\n | | | | public final \n | | Method: wait\n | | | Exceptions: java.lang.InterruptedException \n | | | Parameters: long \n | | | Return Type: void\n | | | Modifiers:\n | | | | public final native \n | | Method: equals\n | | | Exceptions: None\n | | | Parameters: java.lang.Object \n | | | Return Type: boolean\n | | | Modifiers:\n | | | | public \n | | Method: toString\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.String\n | | | Modifiers:\n | | | | public \n | | Method: hashCode\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: int\n | | | Modifiers:\n | | | | public native \n | | Method: getClass\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.Class\n | | | Modifiers:\n | | | | public final native \n | | Method: clone\n | | | Exceptions: java.lang.CloneNotSupportedException \n | | | Parameters: None\n | | | Return Type: java.lang.Object\n | | | Modifiers:\n | | | | protected native \n | | Method: notify\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public final native \n | | Method: notifyAll\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | public final native \n | | Method: registerNatives\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: void\n | | | Modifiers:\n | | | | private static native \n | Constructor: java.lang.Object\n | | Parameters: None\n | | Modifiers:\n | | | public\n========================================\n\nDeclared Class: MyI\n | Methods: \n | | Method: anotherThing\n | | | Exceptions: None\n | | | Parameters: None\n | | | Return Type: java.lang.String\n | | | Modifiers:\n | | | | public abstract \n========================================\n";

+

+ String c1aIr\_printout = "";

+ String c1bIr\_printout = "";

+ String c2Ir\_printout = "";

+

+

+}

diff --git a/src/MyI.java b/src/MyI.java

new file mode 100644

index 0000000..bb06688

--- /dev/null

+++ b/src/MyI.java

@@ -0,0 +1,3 @@

+public interface MyI {

+ public String anotherThing();

+ }

\ No newline at end of file

diff --git a/src/C2.java b/src/C2.java

new file mode 100644

index 0000000..50a58ce

--- /dev/null

+++ b/src/C2.java

@@ -0,0 +1,15 @@

+public class C2 extends C1 implements MyI {

+ public int[] c = new int[2];

+ public final double d = 1.1;

+ public C2() {

+ for (int i = 0; i < c.length; i++) {

+ c[i] = i;

+ }

+ }

+ public void aThing() {

+ }

+ @Override

+ public String anotherThing() {

+ return null;

+ }

+ }

\ No newline at end of file

diff --git a/src/C1.java b/src/C1.java

new file mode 100644

index 0000000..5db2737

--- /dev/null

+++ b/src/C1.java

@@ -0,0 +1,11 @@

+public class C1 {

+ public int a = 0;

+ private long x = 3;

+ public C1() {

+ }

+ public C1(int newA) {

+ a = newA;

+ }

+ public void aThing() {

+ }

+ }

\ No newline at end of file