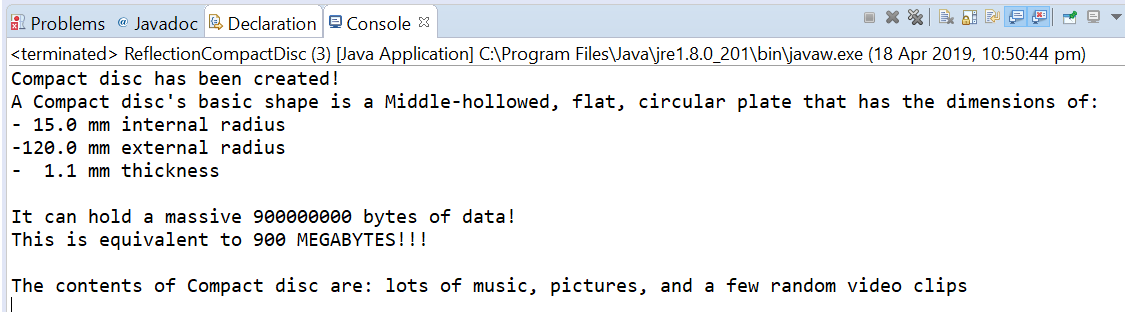
**Task1 – Reflection API**

**Output:**



**Created Program:** **ReflectionCompactDisc.java**

|  |
| --- |
| **package** Task1;  **package** Task1;  **import** java.io.\*;  **import** java.lang.reflect.\*;  **import** java.net.\*;  **import** java.util.\*;  **import** javax.tools.\*;  **public** **class** ReflectionCompactDisc {    **private** **static** String *fieldCapacity*;  **private** **static** String *fieldLabels*;  **private** **static** String *fieldContents*;  **private** **static** String *fieldNameA*;  **private** **static** String *fieldName*;  **private** **static** String *methods*;  **private** **static** String *classRemove* = "Task1.CompactDisc.";  **public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException, InstantiationException, IOException {  String compactDiscSrc = *createCompactDisc*();  //System.out.println(compactDiscSrc);    /\* create the source file \*/  File sourceFile = **new** File("Task1/CompactDisc.java");    **if** (sourceFile.getParentFile().exists() || sourceFile.getParentFile().mkdirs()) {  **try** {  Writer writer = **null**;  **try** {  writer = **new** FileWriter(sourceFile);  writer.write(compactDiscSrc);  writer.flush();  } **finally** {  **try** {  writer.close();  } **catch** (Exception e) {  }  }    /\* Set the environment and compile \*/  System.*setProperty*("java.home", "C:\\Program Files\\Java\\jdk1.8.0\_211\\jre");  DiagnosticCollector<JavaFileObject> diagnostics = **new** DiagnosticCollector<JavaFileObject>();  JavaCompiler compiler = ToolProvider.*getSystemJavaCompiler*();  StandardJavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, **null**, **null**);  /\* This sets up the class path that the compiler will use.\*/  Iterable<? **extends** JavaFileObject> compilationUnit  = fileManager.getJavaFileObjectsFromFiles(Arrays.*asList*(sourceFile));  JavaCompiler.CompilationTask task = compiler.getTask(  **null**,  fileManager,  diagnostics,  **null**,  **null**,  compilationUnit);    /\* Load the class and run \*/  **if** (task.call()) {  URLClassLoader classLoader = **new** URLClassLoader(**new** URL[]{**new** File("./").toURI().toURL()});  Class<?> loadedClass = Class.*forName*("Task1.CompactDisc", **true**, classLoader);  Method meth = loadedClass.getMethod("main", String[].**class**);  String[] params = **null**;  meth.invoke(**null**, (Object) params);    } **else** {  **for** (Diagnostic<? **extends** JavaFileObject> diagnostic : diagnostics.getDiagnostics()) {  System.***out***.format("Error on line %d in %s%n",  diagnostic.getLineNumber(),  diagnostic.getSource().toUri());  }  }  fileManager.close();  } **catch** (IOException | ClassNotFoundException | IllegalAccessException exp) {  exp.printStackTrace();  }  }  }  @SuppressWarnings("rawtypes")  **private** **static** String createCompactDisc() {    String className = "CompactDisc";  Class<CompactDisc> cd = CompactDisc.**class**;  StringBuilder sb = **new** StringBuilder();    String Task1 = "Task1.";  String retrievedMethod = "";    /\* Get all constructors \*/  Constructor[] allCons = cd.getDeclaredConstructors();    /\* Get all methods \*/  Method[] method = cd.getDeclaredMethods();  List<Method> methodArr = (List<Method>) Arrays.*asList*(method);    /\* Get all fields \*/  Field[] allFields = cd.getDeclaredFields();  List<Field> fields = (List<Field>) Arrays.*asList*(allFields);    /\* Create main class \*/  *fieldNameA* = *getFieldName*(fields, "NAME") + "=\"Compact disc\";";    *fieldCapacity* = *getFieldName*(fields, "CAPACITY\_MB") + "=900L;";    *fieldLabels* = *getFieldName*(fields, "label") + ";";    *fieldContents* = *getFieldName*(fields, "contents") + ";";    sb.append("package Task1; \npublic class CompactDisc extends ComputerDisc implements IDataStorage {\n" +  "\n" + *fieldNameA* + "\n" + *fieldCapacity* + "\n" + *fieldLabels* + "\n" + *fieldContents* + "\n");    /\* Create CompactDisc constructor \*/  sb.append("\n" + allCons[0].toString().replace(Task1, "") + "{"  + "\n\t" + "super(120.0D, 15.0D, 1.1D);" + "\n\tcontents = \"\";" + "\n\tlabel = \"[no-label]\";" + "\n}");    /\* Create CompactDisc constructor with parameter \*/  sb.append("\n" + allCons[1].toString().replace(Task1, "").replace("java.lang.String,java.lang.String)", "String l, String c){") +  "\n\tsuper(120.0D, 15.0D, 1.1D);" + "\n\tcontents =c; \n\tlabel =l;" + "\n}");    /\* getLabel method \*/  sb.append("\n" + *getMethod*(methodArr, "getLabel") + "return label;\n}");    /\* setLabel method \*/  retrievedMethod = *getMethod*(methodArr, "setLabel").replace("String", "String l");  sb.append("\n" + retrievedMethod + "label=l;\n}");    /\* setContents method\*/  retrievedMethod = *getMethod*(methodArr, "setContents").replace("String", "String c");  sb.append("\n" + retrievedMethod + "contents = c;\n}");    /\* getContents method\*/  sb.append("\n" + *getMethod*(methodArr, "getContents") + "return contents;\n}");  /\* getCapacity method\*/  sb.append("\n" + *getMethod*(methodArr, "getCapacity") + "return 900000000L;\n}");  /\* getShape method\*/  sb.append("\n" + *getMethod*(methodArr, "getShape") + "return \"Middle-hollowed, flat, circular plate\";\n}");  /\* getName method\*/  sb.append("\n" + *getMethod*(methodArr, "getName") + "return \"Compact disc\";\n}");    /\* getThickness method\*/  sb.append("\n" + *getMethod*(methodArr, "getThickness") + "return 1.1;\n}");  /\* getExternalRadius method\*/  sb.append("\n" + *getMethod*(methodArr, "getExternalRadius") + "return 120.0;\n}");  /\* getInternalRadius method\*/  sb.append("\n" + *getMethod*(methodArr, "getInternalRadius") + "return 15.0;\n}");    /\* Create main method \*/  sb.append("\n" + method[0].toString().replace(*classRemove*,"").replace("java.lang.String[]","String[] args") + "{\n\t" + className + " cd" + "=new " + className + "();" +  "\n\tSystem.out.format(\"%s has been created!\\nA %s's basic shape is a %s that has the dimensions of:\\n-%5.1f mm internal radius\\n-%5.1f mm" +  " external radius\\n-%5.1f mm thickness\\n\", new Object[] {" + " \n\t\tcd." + method[1].getName().toString() + "()," +  " cd." + method[1].getName().toString() + "()," + " cd." + *getMethodName*(method, "getShape") + "()," +  " Double.valueOf(cd." + *getMethodName*(method, "getInternalRadius") + "())," + " Double.valueOf(cd." + *getMethodName*(method, "getExternalRadius") + "())," +  " Double.valueOf(cd." + *getMethodName*(method, "getThickness") + "())});" +  " \n\n\tSystem.out.format(\"\\nIt can hold a massive %d bytes of data!\\nThis is equivalent to %d MEGABYTES!!!\\n\", new Object[] {\n" +  " \t\tLong.valueOf(cd." + *getMethodName*(method, "getCapacity") + "()), Long.valueOf(cd." +  *getMethodName*(method, "getCapacity") + "() / 1000L / 1000L) });" +  "\n\n\tcd." + *getMethodName*(method, "setContents") + "(\"lots of music, pictures, and a few random video clips\");" +  "\n\tSystem.out.format(\"\\nThe contents of %s are: %s\\n\", new Object[] {cd." +  *getMethodName*(method, "getName") + "(), cd." + *getMethodName*(method, "getContents") + "() });");    /\* Add footer }} \*/  sb.append("\n }" + "\n}");  **return** sb.toString();  }    **private** **static** String getMethodName(Method[] method, String methName) {    String methodName = "";  **for** (Method m : method) {  **if**(m.getName().equals(methName)) {  methodName = m.getName();  }  }  **return** methodName;  }    **private** **static** String getFieldName(List<Field> fields, String fieldNam) {    *fieldName* = "";  fields.stream()  .filter(f -> f.getName().equals(fieldNam))  .forEach(f -> *fieldName* = f.toString().replace(*classRemove*,""));    **return** *fieldName*;  }    **private** **static** String getMethod(List<Method> methodArr, String methName) {    *methods* = "";  String methodRemove = "java.lang.";  methodArr.stream()  .filter(m -> m.getName().equals(methName))  .forEach(m -> *methods* = m.toString().replace(methodRemove,"").replace("Task1.", "").replace("CompactDisc.",""));    **return** *methods* + "{\n";  }  } |