University of Arizona, Department of Computer Science

CSc 620 — Assignment 1 — Due midnight, Mon Sep 12 — 5%

Christian Collberg
August 29, 2005

1 Introduction

The purpose of this assignment is to learn Java bytecode, the Jasmin Java assembler, the BCEL bytecode editing library, and the JDI Java debugging library.

You can do this assignment in teams of two.

2 Jasmin

The purpose of this task is to gain some experience writing in Java bytecode. Jasmin can be found here: `http://sourceforge.net/projects/jasmin`.

1. Translate the following Java program `C.java` to a Jasmin assembly file `C.jasmin`:

```
class C {
   static void P(int i) {
      System.out.println(i);
   }

   public static void main (String[] args) {
        for(int i=0; i<10; i++) {
            P(i*100);
        }
   }
}
```

2. Assemble `C.jasmin` using Jasmin into `C.class` and execute it. Debug until it works. `<-;`.
3. Disassemble `C.class` using `javap`.

3 BCEL

The purpose of this task is to gain some experience manipulating Java class files. You can download and read about BCEL here: `http://jakarta.apache.org/bcel`.

Write a BCEL program `Instrument.java` that traces method calls, i.e. it should

1. Open up a Java classfile `C.class` given on the command line:

   ```
   > java Instrument C.class C1.class
   ```

2. Insert, at the beginning and end of every method, code that prints out the name of the method and the values of the incomming arguments:

   ```
   class C {
      static void P(int i) {
         System.out.println("ENTERING C.P");
         System.out.println("    i=",i);
         System.out.println(i);
         System.out.println("EXITING P");
      }

      public static void main (String[] args) {
         System.out.println("ENTERING C.main");
          for(int i=0; i<10; i++) {
              P(i*100);
          }
         System.out.println("EXITING C.main");
      }
   }
   ```

   To simplify things, you only have to print out values of integer arguments.

3. Save the modified classfile to `C1.class`, also give on the command line.

4. Try it out on `C.class` from the previous section.

   ```
   > java Instrument C.class C1.class
   > java C1.class
   ```

   Disassemble `C.class` with `javap` to see that you generated the expected code.

4 JDI

The purpose of this task is to gain some experience executing a Java program under the Java debugger interface, JDI.

The JDI API can be found here: `http://java.sun.com/products/jpda/doc/jdi/index.html`

Write a JDI program `Trace.java` that traces method calls, i.e. it should

1. Open up a Java classfile `C.class` given on the command line:

   ```
   > java -classpath .:tools.jar Trace C.class
   ```

   `tools.jar` comes with Sun's Java distribution.

2. Start running that program under JDI. This is *approximately* what needs to be done to get started:

```
private com.sun.jdi.connect.Connector findConnector(String name) {
   java.util.List connectors =
      com.sun.jdi.Bootstrap.virtualMachineManager().allConnectors();
   java.util.Iterator iter = connectors.iterator();
   while (iter.hasNext()) {
      com.sun.jdi.connect.Connector connector =
         (com.sun.jdi.connect.Connector)iter.next();
      if (connector.name().equals(name)) {
         return connector;
      }
   }
   return null;
}

com.sun.jdi.VirtualMachine vm;
public static void main (String[] args) {

   int traceFlags = com.sun.jdi.VirtualMachine.TRACE_ALL;
   String connectSpec = "com.sun.jdi.CommandLineLaunch";
   com.sun.jdi.connect.Connector connector = findConnector(connectSpec);

   java.util.Map connectorArgs = connector.defaultArguments();
   com.sun.jdi.connect.Connector.Argument argument =
      (com.sun.jdi.connect.Connector.Argument)connectorArgs.get("main");

   String cmdLine = "";
   for (int i=0; i < argv.length; i++)
      cmdLine += argv[i] + " ";
   argument.setValue(cmdLine);

   com.sun.jdi.connect.LaunchingConnector launcher =
      (com.sun.jdi.connect.LaunchingConnector)connector;

   vm = launcher.launch(connectorArgs);
   vm.setDebugTraceMode(traceFlags);

   java.lang.Process process = vm.process();
   vm.resume();
}
```

3. For every method entry and exit point, print out what method was entered, and exited. This can either be done by setting breakpoints (probably the wrong thing to do) or by just listening to the events that JDI generates on entry and exit to every method. Again, this approximates what you need to do:

```
com.sun.jdi.event.EventQueue queue = vm.eventQueue();
try {
     com.sun.jdi.event.EventSet eventSet = queue.remove();
     com.sun.jdi.event.EventIterator it = eventSet.eventIterator();
     while (it.hasNext()) {
          com.sun.jdi.event.Event event = it.nextEvent();
          if (event instanceof com.sun.jdi.event.MethodEntryEvent) {
               ...
```

```
            } else if (event instanceof com.sun.jdi.event.BreakpointEvent) {
                ...
            } else if (event instanceof com.sun.jdi.event.MethodExitEvent) {
                ...
            }
            eventSet.resume();
        }
    } catch (java.lang.InterruptedException exc) {
    } catch (com.sun.jdi.VMDisconnectedException discExc) {}
```

For every method entry you should also print the values of incoming arguments.

4. Try it out on `C.class` from the previous section.

5 Submission and Assessment

The deadline for this assignment is midnight, Mon Sep 12. It is worth 5% of your final grade.

You should submit the assignmen electronically using the `Unix` command

```
turnin cs620.1 README C.jasmin Instrument.java Trace.java
```
.

`README` should briefly describe your implementation and list the members of your team.

Don't show your code to anyone, don't read anyone else's code, don't discuss the details of your code with anyone. If you need help with the assignment see the instructor.