

LIITE 3. Mittausohjelmiston lähdekoodi

Timo Tuominen

Tiedostot

NativeBenchmark	3
Java-komponentit	3
ApplicationState.java	3
ApplicationStateListener.java	4
BenchmarkController.java	4
Benchmark.java	7
BenchmarkParameter.java	8
BenchmarkRegistry.java	14
BenchmarkResult.java	15
BenchmarkRunner.java	17
BenchmarkSelector.java	30
Init.java	38
L.java	39
LogAccess.java	39
measuringtool/AllocatingBenchmarkLongRunningWrapper.java	41
measuringtool/AllocatingBenchmarkShortRunningWrapper.java	42
measuringtool/AllocatingBenchmarkWrapper.java	43
measuringtool/BasicOption.java	43
measuringtool/CommandlineTool.java	45
measuringtool/JavaSystemNanoResponseTimeRecorder.java	47
measuringtool/LinuxPerfRecordTool.java	48
measuringtool/MeasuringOption.java	50
measuringtool/MeasuringTool.java	50
measuringtool/MockCommandlineTool.java	56
measuringtool/OptionSpec.java	57
measuringtool/PlainRunner.java	58
measuringtool/ResponseTimeRecorder.java	59
measuringtool/RunningWrapper.java	60
MockObject.java	61
ShellEnvironment.java	62
SocketCommunicator.java	64
ToolConfig.java	69
Utils.java	73
Python-komponentit (koodingenerointi)	76
benchmark_generator.py	76
jni_types.py	84
make_benchmarks.py	89
make_custom_benchmarks.py	91
templates/arrays.py	103

templates/c_jni_function.py	104
templates/c_module.py	105
templates/c_nativemethod.py	107
templates/__init__.py	109
templates/java_benchmark.py	109
templates/java_counterparts.py	111
templates/java_registry_init.py	112
templates/loop_code.py	113
templating.py	115
Python-komponentit (analyysi)	117
analysis.py	117
datafiles.py	118
gnuplot.py	121
plot_data.py	128
textualtable.py	151

NativeBenchmark

Java-komponentit

Hakemistossa `src/fi/helsinki/cs/tituomin/nativebenchmark/`.

ApplicationState.java

```
1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.content.res.Resources;
4
5 public interface ApplicationState {
6     public void updateState(State state);
7
8     public void updateState(State state, String message);
9
10    public boolean userWantsToRetry(Exception exception);
11
12    public DetailedState getState();
13
14    public Resources getResources();
15
16    public static enum State {
17        INITIALISED(R.string.app_name),
18        MEASURING_STARTED(R.string.measuring_started),
19        INTERRUPTING(R.string.interrupting),
20        INTERRUPTED(R.string.interrupted),
21        MILESTONE(R.string.measuring_milestone),
22        ERROR(R.string.error),
23        INIT_FAIL(R.string.error),
24        MEASURING_FINISHED(R.string.measuring_finished);
25
26        public final int stringId;
27
28        State(int stringId) {
29            this.stringId = stringId;
30        }
31    }
32
33    public class DetailedState {
34        public State state;
35        public String message;
36        private ApplicationState parent;
37
38        public DetailedState(ApplicationState parent) {
39            this.parent = parent;
40            this.state = null;
41            this.message = null;
```

```

42     }
43
44     public DetailedState(ApplicationState parent, DetailedState d) {
45         this.parent = parent;
46         this.state = d.state;
47         this.message = d.message;
48     }
49
50     public String toString() {
51         Resources resources = parent.getResources();
52         if (this.state == null) {
53             return "<unknown state>";
54         }
55         String type = resources.getString(this.state.stringId);
56         return String.format(
57             "%s%s", type,
58             (this.message != null) ? " " + this.message : "");
59     }
60 }
61 }

```

ApplicationStateListener.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 public interface ApplicationStateListener {
4
5     public void stateUpdated(ApplicationState.DetailedState state);
6 }

```

BenchmarkController.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.app.ActivityManager;
4 import android.content.Context;
5 import android.content.res.Resources;
6 import android.os.PowerManager;
7 import android.util.Log;
8
9 import java.io.File;
10 import java.io.IOException;
11
12 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;

```

```

13
14 public class BenchmarkController implements ApplicationState {
15
16     // todo: state changing things synchronized
17
18     public BenchmarkController(Context aContext, File dataDir) {
19         this.detailedState = new ApplicationState.DetailedState(this);
20         this.dataDir = dataDir;
21         this.listeners = new Listeners();
22
23         PowerManager pm = (PowerManager) aContext.getSystemService(Context
24             .POWER_SERVICE);
25         wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
26             "Benchmarking");
27         ActivityManager am = (ActivityManager) aContext.getSystemService
28             (Context.ACTIVITY_SERVICE);
29         this.resources = aContext.getResources();
30         int memoryClass = am.getLargeMemoryClass();
31         Log.v("Selector", "Memory size " + Runtime.getRuntime().maxMemory());
32         Log.v("onCreate", "memoryClass:" + Integer.toString(memoryClass));
33
34     }
35
36     public Resources getResources() {
37         return this.resources;
38     }
39
40     public void updateState(ApplicationState.State state) {
41         updateState(state, null);
42     }
43
44     public void updateState(ApplicationState.State state, String message) {
45         synchronized (this) {
46             this.detailedState.state = state;
47             this.detailedState.message = message;
48
49             switch (state) {
50                 case MEASURING_STARTED:
51                     try {
52                         LogAccess.start(dataDir);
53                     } catch (IOException e) {
54                         this.detailedState.state = ApplicationState.State.ERROR;
55                         this.detailedState.message = "Could not initialize " +
56                             "log file.";
57                         Log.e(TAG, this.detailedState.message);
58                     }
59                     wakeLock.acquire();
60                     break;
61                 case ERROR:
62                 case INTERRUPTED:
63                 case MEASURING_FINISHED:
64                     if (wakeLock.isHeld()) {
65                         wakeLock.release();

```

```

66         }
67         LogAccess.end();
68         case INITIALISED:
69         case INIT_FAIL:
70         case MILESTONE:
71             if (this.listeners.milestoneListener != null) {
72                 this.listeners.milestoneListener.stateUpdated(this
73                     .detailedState);
74             }
75         }
76     }
77 }
78
79 public ApplicationState.DetailedState getState() {
80     synchronized (this) {
81         return new ApplicationState.DetailedState(this, this.detailedState);
82     }
83 }
84
85 public boolean userWantsToRetry(Exception e) {
86     return false; // TODO: make interaction configurable
87 }
88
89 private class BenchRunnable implements Runnable {
90     BenchRunnable(BenchmarkRunner runner, ToolConfig configuration) {
91         this.runner = runner;
92         this.configuration = configuration;
93     }
94
95     public void run() {
96         this.runner.runBenchmarks(
97             BenchmarkController.this,
98             this.configuration,
99             dataDir);
100     }
101
102     private BenchmarkRunner runner;
103     private ToolConfig configuration;
104 }
105
106 public void startMeasuring(BenchmarkRunner runner, ToolConfig
107     configuration) {
108     String message = null;
109     if (this.resources.getString(R.string.app_dirty).equals("1")) {
110         message = this.resources.getString(R.string.warning_changed);
111     }
112
113     BenchmarkRunner.BenchmarkSet set = configuration.getBenchmarkSet();
114     runner.setBenchmarkSet(set);
115     measuringThread = new Thread(new BenchRunnable(runner, configuration));
116     if (message != null) {
117         this.updateState(ApplicationState.State.MEASURING_STARTED, message);
118     } else {

```

```

119         this.updateState(ApplicationState.State.MEASURING_STARTED);
120     }
121     measuringThread.start();
122 }
123
124 public void interruptMeasuring() {
125     MeasuringTool.userInterrupt();
126     measuringThread.interrupt();
127 }
128
129 public void addListener(ApplicationStateListener listener,
130                        ApplicationState.State state) {
131     if (state == ApplicationState.State.MILESTONE) {
132         this.listeners.milestoneListener = listener;
133     }
134 }
135
136 public void removeListeners() {
137     this.listeners = null;
138 }
139
140 private class Listeners {
141     public ApplicationStateListener milestoneListener;
142
143     public Listeners() {
144         milestoneListener = null;
145     }
146 }
147
148 private Listeners listeners;
149 private ApplicationState.State state;
150 private String message;
151 private ApplicationState.DetailedState detailedState;
152 private PowerManager.WakeLock wakeLock;
153 private File dataDir;
154 private Thread measuringThread;
155 private static final String TAG = "BenchmarkController";
156 private Resources resources;
157
158 }

```

Benchmark.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.os.Process;
4
5 public abstract class Benchmark implements Runnable {
6     protected abstract void runInternal();

```

```

7
8     public abstract String from();
9
10    public abstract String to();
11
12    public abstract String description();
13
14    public abstract String id();
15
16    public abstract int sequenceNo();
17
18    public abstract boolean isAllocating();
19
20    public abstract boolean isNonvirtual();
21
22    public abstract boolean dynamicParameters();
23
24    public abstract boolean representative();
25
26    public long repetitionsLeft;
27
28    public void run() {
29        Process.setThreadPriority(-5);
30        runInternal();
31    }
32
33    protected BenchmarkParameter benchmarkParameter;
34    protected long repetitions;
35
36    public void init(BenchmarkParameter bp) {
37        repetitionsLeft = 0;
38        benchmarkParameter = bp;
39        repetitions = -1;
40    }
41
42    public void setRepetitions(long reps) {
43        if (reps < 1) {
44            return;
45        }
46        repetitions = reps;
47        BenchmarkRegistry.setRepetitions(reps);
48    }
49 }

```

BenchmarkParameter.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.content.pm.PermissionInfo;

```



```

4 import android.util.Log;
5
6 import java.nio.ByteBuffer;
7 import java.util.Arrays;
8 import java.util.Iterator;
9 import java.util.NoSuchElementException;
10
11 import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.JavaCounterparts;
12
13 // Important! Imported as an example class.
14
15 public class BenchmarkParameter implements Iterable<Integer> {
16
17     private native int initReturnvalues(int size, MockObject o);
18
19     private native void freeReturnvalues();
20
21     public void setUp() {
22         initReturnvalues(index * DEFAULTSIZE, mockObjectInstance);
23         JavaCounterparts.initParams(this);
24     }
25
26     public void tearDown() {
27         freeReturnvalues();
28     }
29
30     public static final int DEFAULTSIZE = 64;
31     public static final int RANGE = 8;
32     public static final int MAXSIZE = DEFAULTSIZE * RANGE;
33
34     // The byte buffer has to hold longs and doubles too.
35     public static final int NIO_MAXSIZE = DEFAULTSIZE * RANGE * 8;
36
37     public static MockObject mockObjectInstance = new MockObject();
38
39     public MockObject retrieveMockObject() {
40         return mockObjectInstance;
41     }
42
43     public BenchmarkParameter() {
44         index = 1;
45         if (!generated) {
46             generateAll();
47             generated = true;
48         }
49
50         for (int i = 0; i < RANGE + 1; i++) {
51             if (STRINGS[i] == null) {
52                 Log.v("Parameter", "STRINGS is null at " + i);
53             }
54             if (OBJECTS[i] == null) {
55                 Log.v("Parameter", "OBJECTS is null at " + i);
56             }

```

```

57         if (THROWABLES[i] == null) {
58             Log.v("Parameter", "THROWABLES is null at " + i);
59         }
60         if (BOOLEAN_ARRAYS[i] == null) {
61             Log.v("Parameter", "BOOLEAN_ARRAYS is null at " + i);
62         }
63         if (BYTE_ARRAYS[i] == null) {
64             Log.v("Parameter", "BYTE_ARRAYS is null at " + i);
65         }
66         if (CHAR_ARRAYS[i] == null) {
67             Log.v("Parameter", "CHAR_ARRAYS is null at " + i);
68         }
69         if (DOUBLE_ARRAYS[i] == null) {
70             Log.v("Parameter", "DOUBLE_ARRAYS is null at " + i);
71         }
72         if (FLOAT_ARRAYS[i] == null) {
73             Log.v("Parameter", "FLOAT_ARRAYS is null at " + i);
74         }
75         if (INT_ARRAYS[i] == null) {
76             Log.v("Parameter", "INT_ARRAYS is null at " + i);
77         }
78         if (LONG_ARRAYS[i] == null) {
79             Log.v("Parameter", "LONG_ARRAYS is null at " + i);
80         }
81         if (SHORT_ARRAYS[i] == null) {
82             Log.v("Parameter", "SHORT_ARRAYS is null at " + i);
83         }
84         if (OBJECT_ARRAYS[i] == null) {
85             Log.v("Parameter", "OBJECT_ARRAYS is null at " + i);
86         }
87     }
88 }
89
90
91 // must call initreturnvalues and freereturnvalues after...
92 public void setIndex(int index) {
93     if (index < (RANGE + 1)) {
94         this.index = index;
95     } else {
96         throw new IllegalArgumentException("Requested size too large. " +
97             index);
98     }
99 }
100
101 public int getIndex() {
102     return this.index;
103 }
104
105 public int getSize() {
106     return DEFAULTSIZE * index;
107 }
108
109

```

```

110 public Iterator<Integer> iterator() {
111     return new RangeIterator();
112 }
113
114 private class RangeIterator implements Iterator<Integer> {
115     private int index;
116
117     public RangeIterator() {
118         index = -1;
119     }
120
121     public boolean hasNext() {
122         return index < RANGE;
123     }
124
125     public Integer next() {
126         if (!hasNext()) {
127             throw new NoSuchElementException();
128         }
129         index++;
130         setIndex(index);
131         return DEFAULTSIZE * index;
132     }
133
134     public void remove() {
135         throw new UnsupportedOperationException();
136     }
137 }
138
139 public boolean[] retrieveBooleanArray() {
140     return BOOLEAN_ARRAYS[index];
141 }
142
143 public byte[] retrieveByteArray() {
144     return BYTE_ARRAYS[index];
145 }
146
147 public char[] retrieveCharArray() {
148     return CHAR_ARRAYS[index];
149 }
150
151 public double[] retrieveDoubleArray() {
152     return DOUBLE_ARRAYS[index];
153 }
154
155 public float[] retrieveFloatArray() {
156     return FLOAT_ARRAYS[index];
157 }
158
159 public int[] retrieveIntArray() {
160     return INT_ARRAYS[index];
161 }
162

```

```

163
164 public long[] retrieveLongArray() {
165     return LONG_ARRAYS[index];
166 }
167
168 public short[] retrieveShortArray() {
169     return SHORT_ARRAYS[index];
170 }
171
172 public Object[] retrieveObjectArray() {
173     return OBJECT_ARRAYS[index];
174 }
175
176 public Object retrieveObject() {
177     return OBJECTS[index];
178 }
179
180 public Class retrieveClass() {
181     // todo: causes bump in measurements
182     return OBJECTS[index].getClass();
183 }
184
185 public String retrieveString() {
186     String ret = STRINGS[index];
187     return ret;
188 }
189
190 public Throwable retrieveThrowable() {
191     return THROWABLES[index];
192 }
193
194 public ByteBuffer retrieveDirectByteBuffer() {
195     return BYTEBUFFER;
196 }
197
198 // -----
199
200 private static void generateAll() {
201     int index = RANGE - 1;
202     int size = MAXSIZE - DEFAULTSIZE;
203     generateMax();
204     while (index > -1) {
205         BOOLEAN_ARRAYS[index] = Arrays.copyOf(BOOLEAN_ARRAYS[RANGE], size);
206         CHAR_ARRAYS[index] = Arrays.copyOf(CHAR_ARRAYS[RANGE], size);
207         BYTE_ARRAYS[index] = Arrays.copyOf(BYTE_ARRAYS[RANGE], size);
208         INT_ARRAYS[index] = Arrays.copyOf(INT_ARRAYS[RANGE], size);
209         LONG_ARRAYS[index] = Arrays.copyOf(LONG_ARRAYS[RANGE], size);
210         SHORT_ARRAYS[index] = Arrays.copyOf(SHORT_ARRAYS[RANGE], size);
211         DOUBLE_ARRAYS[index] = Arrays.copyOf(DOUBLE_ARRAYS[RANGE], size);
212         FLOAT_ARRAYS[index] = Arrays.copyOf(FLOAT_ARRAYS[RANGE], size);
213
214         OBJECT_ARRAYS[index] = Arrays.copyOf(OBJECT_ARRAYS[RANGE], size);
215

```

```

216         OBJECTS[index] = OBJECT;
217         THROWABLES[index] = THROWABLE;
218         STRINGS[index] = STRING_BUILDER.substring(0, size);
219
220         index--;
221         size -= DEFAULTSIZE;
222     }
223 }
224
225 private static void generateMax() {
226     char c = 0;
227     boolean b = true;
228     byte by = 0;
229     int v = 0;
230     long l = 0;
231     short s = 0;
232     double d = 0;
233     float f = 0;
234
235     BOOLEAN_ARRAYS[RANGE] = new boolean[MAXSIZE];
236     CHAR_ARRAYS[RANGE] = new char[MAXSIZE];
237     BYTE_ARRAYS[RANGE] = new byte[MAXSIZE];
238     INT_ARRAYS[RANGE] = new int[MAXSIZE];
239     LONG_ARRAYS[RANGE] = new long[MAXSIZE];
240     SHORT_ARRAYS[RANGE] = new short[MAXSIZE];
241     DOUBLE_ARRAYS[RANGE] = new double[MAXSIZE];
242     FLOAT_ARRAYS[RANGE] = new float[MAXSIZE];
243     OBJECT_ARRAYS[RANGE] = new Object[MAXSIZE];
244
245     for (int i = 0; i < MAXSIZE; i++) {
246         STRING_BUILDER.append(c);
247
248         BOOLEAN_ARRAYS[RANGE][i] = b;
249         CHAR_ARRAYS[RANGE][i] = c;
250         BYTE_ARRAYS[RANGE][i] = by;
251         INT_ARRAYS[RANGE][i] = v;
252         LONG_ARRAYS[RANGE][i] = l;
253         SHORT_ARRAYS[RANGE][i] = s;
254         DOUBLE_ARRAYS[RANGE][i] = d;
255         FLOAT_ARRAYS[RANGE][i] = f;
256         OBJECT_ARRAYS[RANGE][i] = OBJECT;
257         OBJECTS[RANGE] = OBJECT;
258         THROWABLES[RANGE] = THROWABLE;
259
260         b = !b;
261         by = (byte) ((by + 1) % Byte.MAX_VALUE);
262         v = (v + 1) % Integer.MAX_VALUE;
263         l = (l + 1) % Long.MAX_VALUE;
264         s = (short) ((s + 1) % Short.MAX_VALUE);
265         d = (d + 0.1);
266         f = (f + 0.1f);
267         c = (char) ((char) (c + '\u0001') % (char) Character.MAX_VALUE);
268     }

```

```

269     STRINGS[RANGE] = STRING_BUILDER.substring(0, MAXSIZE);
270 }
271
272 private int index;
273 private static boolean generated = false;
274
275 private static final StringBuilder STRING_BUILDER = new StringBuilder();
276 private static final Object OBJECT = new PermissionInfo();
277 private static final Throwable THROWABLE = new Exception();
278
279 private static final String[] STRINGS = new String[RANGE + 1];
280 private static final Object[] OBJECTS = new PermissionInfo[RANGE + 1];
281 private static final Throwable[] THROWABLES = new Exception[RANGE + 1];
282
283 private static final boolean[][] BOOLEAN_ARRAYS = new boolean[RANGE + 1][];
284 private static final byte[][] BYTE_ARRAYS = new byte[RANGE + 1][];
285 private static final char[][] CHAR_ARRAYS = new char[RANGE + 1][];
286 private static final double[][] DOUBLE_ARRAYS = new double[RANGE + 1][];
287 private static final float[][] FLOAT_ARRAYS = new float[RANGE + 1][];
288 private static final int[][] INT_ARRAYS = new int[RANGE + 1][];
289 private static final long[][] LONG_ARRAYS = new long[RANGE + 1][];
290 private static final short[][] SHORT_ARRAYS = new short[RANGE + 1][];
291 private static final Object[][] OBJECT_ARRAYS = new Object[RANGE + 1][];
292
293 private static final ByteBuffer BYTEBUFFER = ByteBuffer.allocateDirect
294     (NIO_MAXSIZE);
295
296 }

```

BenchmarkRegistry.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import java.util.LinkedList;
4 import java.util.List;
5
6 import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.JavaCounterparts;
7
8 public class BenchmarkRegistry {
9
10     private static List<Benchmark> benchmarks;
11
12     public static long repetitions;
13     public static final long CHECK_INTERRUPTED_INTERVAL = 1000;
14
15     public static List<Benchmark> getBenchmarks() {
16         return benchmarks;
17     }
18

```

```

19  public static void init(long reps) throws ClassNotFoundException {
20      repetitions = reps;
21      benchmarks = new LinkedList<Benchmark>();
22      Class jCounterparts = Class.forName("fi.helsinki.cs.tituomin" +
23          ".nativebenchmark.benchmark.JavaCounterparts");
24      Class threadClass = Class.forName("java.lang.Thread");
25      initNative(reps, CHECK_INTERRUPTED_INTERVAL, jCounterparts,
26          JavaCounterparts.INSTANCE, threadClass);
27  }
28
29  public static native void initNative(long repetitions, long interval,
30      Class javaCounterparts,
31      JavaCounterparts counterInstance,
32      Class threadClass);
33
34  public static native void setRepetitions(long repetitions);
35
36  public static native void interruptNative();
37
38  public static native void resetInterruptFlag();
39 }

```

BenchmarkResult.java

```

1  package fi.helsinki.cs.tituomin.nativebenchmark;
2
3  import android.util.Log;
4
5  import java.util.HashMap;
6  import java.util.Map;
7  import java.util.Map.Entry;
8
9
10 public class BenchmarkResult {
11
12     public BenchmarkResult() {
13         values = new String[ESTIMATED_CAPACITY];
14         valueCount = 0;
15     }
16
17     public void put(String label, String value) {
18         Integer labelIndex = labelIndexes.get(label);
19         if (labelIndex == null) {
20             lastIndex++;
21             if (lastIndex > labels.length) {
22                 Log.e("BenchmarkResults", "Error, too many kinds of values, " +
23                     "increase capacity!");
24             }
25             labels[lastIndex] = label;

```

```

26         labelIndexes.put(label, lastIndex);
27         labelIndex = lastIndex;
28     }
29     values[labelIndex] = value;
30     valueCount++;
31 }
32
33 public String get(String label) {
34     Integer index = labelIndexes.get(label);
35     if (index != null) {
36         return values[index];
37     } else {
38         return null;
39     }
40 }
41
42 public String get(int i) {
43     return values[i];
44 }
45
46 public void putAll(BenchmarkResult other) {
47     String[] otherValues = other.getValues();
48     for (int i = 0; i < size(); i++) {
49         if (otherValues[i] != null) {
50             values[i] = otherValues[i];
51             valueCount++;
52         }
53     }
54 }
55
56 public void putAll(Map<String, String> map) {
57     for (Entry<String, String> entry : map.entrySet()) {
58         put(entry.getKey(), entry.getValue());
59     }
60 }
61
62 public String[] getValues() {
63     return values;
64 }
65
66 public static String getLabel(int i) {
67     return labels[i];
68 }
69
70 public static String[] labels() {
71     return labels;
72 }
73
74 public boolean isEmpty() {
75     return (valueCount == 0);
76 }
77
78 public static int size() {

```



```

79         return (lastIndex + 1);
80     }
81
82     private static final int ESTIMATED_CAPACITY = 200;
83     private String[] values;
84     private int valueCount;
85
86     private static String[] labels = new String[ESTIMATED_CAPACITY];
87     private static Map<String, Integer> labelIndexes = new HashMap<String,
88         Integer>(ESTIMATED_CAPACITY);
89     private static int lastIndex = -1;
90 }

```

BenchmarkRunner.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.os.SystemClock;
4 import android.util.Log;
5 import android.util.Pair;
6
7 import java.io.File;
8 import java.io.FileInputStream;
9 import java.io.FileNotFoundException;
10 import java.io.IOException;
11 import java.io.InputStream;
12 import java.io.OutputStream;
13 import java.io.PrintWriter;
14 import java.lang.reflect.Method;
15 import java.lang.reflect.Modifier;
16 import java.util.ArrayList;
17 import java.util.Arrays;
18 import java.util.Collections;
19 import java.util.Date;
20 import java.util.HashMap;
21 import java.util.Iterator;
22 import java.util.List;
23 import java.util.Map;
24 import java.util.zip.ZipEntry;
25 import java.util.zip.ZipOutputStream;
26
27 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
28 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool
29     .RunnerException;
30
31 import static fi.helsinki.cs.tituomin.nativebenchmark.Utills.colPr;
32
33 public enum BenchmarkRunner {
34     INSTANCE; // singleton enum pattern

```

```

35
36 private static final String SEPARATOR = ",";
37 private static final String MISSING_VALUE = "-";
38 private static final long WARMUP_REPS = 50000;
39 private static BenchmarkParameter benchmarkParameter;
40 private static List<MeasuringTool> measuringTools;
41 private static int benchmarkCounter = 0;
42
43 private static boolean interrupted = false;
44
45 public static BenchmarkParameter getBenchmarkParameter() {
46     if (benchmarkParameter == null) {
47         benchmarkParameter = new BenchmarkParameter();
48     }
49     return benchmarkParameter;
50 }
51
52 private BenchmarkRunner() {
53     this.allocatingRepetitions = -1;
54 }
55
56 public void initTools(ToolConfig conf, long repetitions, long
57     allocRepetitions) throws IOException, InterruptedException {
58
59     conf.setDefaultRepetitions(this.repetitions);
60     conf.setDefaultRunAllBenchmarks(this.runAllBenchmarks);
61     if (this.allocatingRepetitions > 0) {
62         conf.setDefaultAllocRepetitions(this.allocatingRepetitions);
63     }
64
65     measuringTools = new ArrayList<MeasuringTool>();
66     for (MeasuringTool tool : conf) {
67         measuringTools.add(tool);
68     }
69 }
70
71 private long repetitions;
72 private long allocatingRepetitions;
73 private CharSequence appRevision;
74 private CharSequence appChecksum;
75 private File cacheDir;
76 private boolean runAllBenchmarks;
77 private boolean runAtMaxSpeed;
78 private String benchmarkSubstring;
79
80 public enum BenchmarkSet {
81     ALLOC,
82     NON_ALLOC
83 }
84
85 ;
86 private BenchmarkSet benchmarkSet;
87

```

```

88     public BenchmarkRunner setRepetitions(long x) {
89         repetitions = x;
90         return this;
91     }
92
93     public BenchmarkRunner setAllocatingRepetitions(long x) {
94         allocatingRepetitions = x;
95         return this;
96     }
97
98     public BenchmarkRunner setAppRevision(CharSequence x) {
99         appRevision = x;
100        return this;
101    }
102
103    public BenchmarkRunner setAppChecksum(CharSequence x) {
104        appChecksum = x;
105        return this;
106    }
107
108    public BenchmarkRunner setCacheDir(File x) {
109        cacheDir = x;
110        return this;
111    }
112
113    public BenchmarkRunner setRunAllBenchmarks(boolean x) {
114        runAllBenchmarks = x;
115        return this;
116    }
117
118    public BenchmarkRunner setRunAtMaxSpeed(boolean x) {
119        runAtMaxSpeed = x;
120        return this;
121    }
122
123    public BenchmarkRunner setBenchmarkSubstring(String x) {
124        benchmarkSubstring = x;
125        return this;
126    }
127
128    public BenchmarkRunner setBenchmarkSet(BenchmarkSet x) {
129        benchmarkSet = x;
130        return this;
131    }
132
133    public void runBenchmarks(ApplicationState mainUI, ToolConfig config,
134                           File dataDir) {
135        interrupted = false;
136
137        try {
138            BenchmarkRegistry.init(this.repetitions);
139            // todo replace with config
140            MeasuringTool.setDataDir(dataDir);

```

```

141         Log.v(TAG, config.toString());
142         initTools(config, this.repetitions, this.allocatingRepetitions);
143     } catch (Exception e) {
144         mainUI.updateState(ApplicationState.State.ERROR);
145         Log.e("BenchmarkRunner", "Error initialising", e);
146         return;
147     }
148
149     benchmarkParameter = getBenchmarkParameter();
150     BenchmarkInitialiser.init(benchmarkParameter);
151
152     List<Benchmark> allBenchmarks = BenchmarkRegistry.getBenchmarks();
153
154     // todo enable
155     long seed = System.currentTimeMillis();
156     Log.i(TAG, String.format("Random seed %d", seed));
157     java.util.Random random = new java.util.Random(seed);
158     Collections.shuffle(allBenchmarks, random);
159     try {
160         Init.initEnvironment(true); // run warmup at max speed
161     } catch (IOException e) {
162         handleException(e, mainUI);
163         return;
164     }
165     for (MeasuringTool tool : measuringTools) {
166         if (tool == null) {
167             return;
168         }
169         if (interrupted) {
170             return;
171         }
172
173         List<Benchmark> benchmarks = new ArrayList<Benchmark>();
174
175         if (this.benchmarkSubstring == null) {
176             this.benchmarkSubstring = "";
177         }
178         String substringToApply = "";
179
180         String filter = tool.getFilter();
181         if (filter != null && !filter.equals("")) {
182             substringToApply = tool.getFilter().toLowerCase();
183         }
184         this.benchmarkSubstring = substringToApply;
185
186         for (Benchmark b : allBenchmarks) {
187             boolean selected;
188             if (tool.runAllBenchmarks()) {
189                 selected = true;
190             } else if (!substringToApply.equals("")) {
191                 selected = (
192                     b.getClass().getSimpleName().toLowerCase().indexOf(
193                         substringToApply) != -1);

```

```

194         } else {
195             selected = (
196                 b.representative() &&
197                 ((!b.isAllocating()) && this.benchmarkSet
198                     == BenchmarkSet.NON_ALLOC) ||
199                 (b.isAllocating() && this.benchmarkSet ==
200                     BenchmarkSet.ALLOC));
201         }
202         if (b.isNonvirtual() &&
203             b.from() != "C" &&
204             b.to() != "J") {
205             if (L.log) {
206                 Log.i(TAG, String.format("skipping nonvirtual %s", b
207                     .id()));
208             }
209             selected = false;
210         }
211         if (selected) {
212             benchmarks.add(b);
213         }
214     }
215     int numOfBenchmarks = benchmarks.size();
216
217     if (L.log) {
218         Log.i(TAG, tool.getClass().getSimpleName());
219     }
220
221     if (!tool.ignore()) {
222         // set the slower CPU frequency etc. after the warmup
223         // round(s), taking less time
224         if (!this.runAtMaxSpeed) {
225             try {
226                 Init.initEnvironment(false);
227             } catch (IOException e) {
228                 handleException(e, mainUI);
229                 return;
230             }
231         }
232     }
233
234     int max_rounds = tool.getRounds();
235     String measurementID = Utils.getUUID();
236     File resultFile = new File(dataDir, "benchmarks-" + measurementID
237         + ".csv");
238     long startTime = SystemClock.uptimeMillis();
239     Date start = new Date();
240     long endTime = 0;
241
242     int round = -1;
243     boolean labelsWritten = false;
244
245     ROUNDLOOP:
246     while (++round < max_rounds) {

```

```

247 benchmarkCounter = 0;
248 PrintWriter tempWriter = null;
249 File tempFile = new File(this.cacheDir, "benchmarks-temp.csv");
250 try {
251     tempWriter = Utils.makeWriter(tempFile, false);
252 } catch (FileNotFoundException e) {
253     handleException(e, mainUI);
254     return;
255 }
256
257
258 List<BenchmarkResult> collectedData;
259
260 try {
261     if (tool.explicitGC()) {
262         System.gc();
263         Thread.sleep(500);
264     }
265 } catch (InterruptedException e) {
266     logE("Measuring thread was interrupted");
267     mainUI.updateState(
268         ApplicationState.State.INTERRUPTED);
269     interrupted = true;
270     break ROUNDLOOP;
271 }
272 int count = benchmarks.size();
273 int j = 0;
274 for (Benchmark benchmark : benchmarks) {
275     if (L.log) {
276         Log.i(TAG, (count - j) + " left");
277         Log.i(TAG, benchmark.getClass().getSimpleName());
278     }
279     ;
280     j++;
281     try {
282         collectedData = runSeries(benchmark, mainUI, tool,
283             round, max_rounds, count, j);
284     } catch (RunnerException e) {
285         logE("Exception was thrown", e.getCause());
286         mainUI.updateState(
287             ApplicationState.State.ERROR);
288         interrupted = true;
289         break ROUNDLOOP;
290     } catch (InterruptedException e) {
291         logE("Measuring thread was interrupted");
292         mainUI.updateState(
293             ApplicationState.State.INTERRUPTED);
294
295         interrupted = true;
296         break ROUNDLOOP;
297     }
298
299     if (collectedData.isEmpty() || tool.ignore()) {

```

```

300         continue;
301     }
302
303     endTime = SystemClock.uptimeMillis();
304
305     // print data
306     for (BenchmarkResult result : collectedData) {
307         for (int i = 0; i < BenchmarkResult.size(); i++) {
308             String value = result.get(i);
309             if (value == null) {
310                 value = MISSING_VALUE;
311             }
312             tempWriter.print(value);
313             tempWriter.print(SEPARATOR);
314         }
315         tempWriter.println("");
316         tempWriter.flush();
317     }
318 }
319 endTime = SystemClock.uptimeMillis();
320 tempWriter.close();
321
322 if (tool.ignore()) {
323     continue; // todo test
324 }
325
326 InputStream in = null;
327 OutputStream out = null;
328 PrintWriter resultWriter = null;
329 try {
330     resultWriter = Utils.makeWriter(resultFile, true);
331     // note: labels should be written last, after
332     // all possible keys have been created
333     if (!labelsWritten) {
334         String[] labels = BenchmarkResult.labels();
335         for (int i = 0; i < BenchmarkResult.size(); i++) {
336             resultWriter.print(labels[i] + SEPARATOR);
337         }
338         resultWriter.println("");
339         resultWriter.close();
340         labelsWritten = true;
341     }
342
343     in = new FileInputStream(tempFile);
344     out = Utils.makeOutputStream(resultFile, true);
345     Utils.copyStream(in, out);
346     //tempFile.delete();
347 } catch (Exception e) {
348     mainUI.updateState(ApplicationState.State.ERROR);
349     Log.e("BenchmarkRunner", "Error writing results", e);
350     interrupted = true;
351     break ROUNDLOOP;
352 } finally {

```

```

353         try {
354             if (in != null) {
355                 in.close();
356             }
357             if (out != null) {
358                 out.flush();
359                 out.close();
360             }
361             if (resultWriter != null) {
362                 resultWriter.close();
363             }
364         } catch (IOException e) {
365             mainUI.updateState(ApplicationState.State.ERROR);
366             Log.e("BenchmarkRunner", "Error closing files", e);
367             interrupted = true;
368             break ROUNDLOOP;
369         }
370     }
371 }
372
373 if (!tool.ignore()) {
374     // there has been at least one succesful round
375     writeMeasurementMetadata(
376         new File(dataDir, "measurements.txt"),
377         measurementID,
378         (this.runAtMaxSpeed ? Init.CPUFREQ_MAX : Init.CPUFREQ),
379         startTime, endTime, start, tool, numOfBenchmarks,
380         round);
381
382     List<String> filenames = tool.getFileNames();
383     if (!filenames.isEmpty()) {
384         OutputStream os = null;
385
386         try {
387             File zip = new File(dataDir, "perfddata-" +
388                 measurementID + ".zip");
389             os = Utils.makeOutputStream(zip, false);
390             Log.v("BenchmarkRunner", "filenames " + filenames
391                 .size());
392             File measurementMetadataFile = new File(
393                 dataDir, "benchmarks-" + measurementID + "" +
394                 ".csv");
395             if (measurementMetadataFile.exists()) {
396                 // TODO: actually make sure csv is written for
397                 // incomplete runs
398                 filenames.add(measurementMetadataFile
399                     .getAbsolutePath());
400             }
401             writeToZipFile(os, filenames, measurementID);
402             deleteFiles(filenames);
403         } catch (FileNotFoundException e) {
404             logE(e);
405         } catch (IOException e) {

```



```

406         logE("Error writing zip file.", e);
407     }
408     try {
409         if (os != null) {
410             os.close();
411         }
412     } catch (IOException e) {
413         logE("Error closing file.", e);
414     }
415 }
416 }
417
418 }
419 mainUI.updateState(
420     ApplicationState.State.MEASURING_FINISHED);
421 }
422
423 private void writeMeasurementMetadata(
424     File catalogFile, String measurementID, int cpuFreq,
425     long startTime, long endTime, Date start, MeasuringTool tool,
426     int numOfBenchmarks, int rounds) {
427
428     Date end = new Date();
429     PrintWriter c = null;
430     try {
431         c = Utils.makeWriter(catalogFile, true);
432
433         c.println("");
434         for (Pair<String, String> pair : tool.configuration()) {
435             colPr(c, pair.first, pair.second);
436         }
437         colPr(c, "id", measurementID);
438         colPr(c, "cpu-freq", cpuFreq);
439         colPr(c, "logfile", LogAccess.filename());
440         colPr(c, "rounds", rounds);
441         colPr(c, "start", start);
442         colPr(c, "end", end);
443         colPr(c, "duration", Utils.humanTime(endTime - startTime));
444         colPr(c, "benchmarks", numOfBenchmarks);
445         colPr(c, "code-revision", this.appRevision);
446         colPr(c, "code-checksum", this.appChecksum);
447         colPr(c, "benchmark-set", this.benchmarkSet);
448         colPr(c, "substring-filter", this.benchmarkSubstring);
449         c.println("");
450     } catch (IOException e) {
451         logE(e);
452     } finally {
453         c.close();
454     }
455 }
456
457 private final static String TAG = "BenchmarkRunner";
458

```

```

459 private static void logE(String message, Throwable e) {
460     Log.e(TAG, message, e);
461 }
462
463 private static void logE(Throwable e) {
464     Log.e(TAG, "exception", e);
465 }
466
467 private static void logE(String msg) {
468     Log.e(TAG, msg);
469 }
470
471 private static void deleteFiles(List<String> filenames) {
472     for (String filename : filenames) {
473         boolean success = new File(filename).delete();
474         if (!success) {
475             logE("Error deleting file " + filename);
476         }
477     }
478 }
479
480 private static void handleException(Exception e, ApplicationState UI) {
481     logE(e);
482     UI.updateState(ApplicationState.State.ERROR);
483     return;
484 }
485
486 private static void
487 writeToZipFile(OutputStream os, List<String> filenames, String mID)
488     throws IOException {
489     ZipOutputStream zos = new ZipOutputStream(os);
490     final int byteCount = 512 * 1024;
491     byte[] bytes = new byte[byteCount];
492     for (String filename : filenames) {
493         try {
494             InputStream is = Utils.makeInputStream(filename);
495             ZipEntry entry = new ZipEntry(mID + "/" + new File(filename)
496                 .getName());
497             int bytesRead = -1;
498             zos.putNextEntry(entry);
499             while ((bytesRead = is.read(bytes, 0, byteCount)) != -1) {
500                 zos.write(bytes, 0, bytesRead);
501             }
502             zos.closeEntry();
503         } finally {
504             zos.flush();
505         }
506     }
507     try {
508         zos.close();
509     } catch (IOException e) {
510         logE(e);
511     }

```

```

512     }
513
514     private static List<BenchmarkResult> runSeries(
515         Benchmark benchmark, ApplicationState mainUI, MeasuringTool tool,
516         int roundNo, int roundCount, int benchmarkCount, int benchmarkIndex)
517         throws InterruptedException, RunnerException {
518
519         List<BenchmarkResult> compiledMetadata = new
520             ArrayList<BenchmarkResult>();
521
522         if (Thread.interrupted()) {
523             throw new InterruptedException();
524         }
525         BenchmarkParameter bPar = getBenchmarkParameter();
526         BenchmarkResult introspected;
527         try {
528             introspected = inspectBenchmark(benchmark);
529         } catch (ClassNotFoundException e) {
530             Log.e("BenchmarkRunner", "Could not find class", e);
531             return compiledMetadata;
532         }
533
534         String refTypesString = introspected.get("has_reference_types");
535         boolean hasRefTypes = (refTypesString != null) && (refTypesString
536             .equals("1"));
537
538         String parameterCountString = introspected.get("parameter_count");
539         int parameterCount = (parameterCountString == null) ? -1 : Integer
540             .parseInt(parameterCountString);
541
542         boolean dynamicParameters =
543             benchmark.dynamicParameters() ||
544             (hasRefTypes && (-1 < parameterCount &&
545                 parameterCount < 2));
546
547         Iterator<Integer> iterator = bPar.iterator();
548         Integer i;
549         int j = -1;
550         if (iterator.hasNext()) {
551             i = iterator.next();
552             j++;
553         } else {
554             i = null;
555         }
556         while (i != null) {
557             if (Thread.interrupted()) {
558                 throw new InterruptedException();
559             }
560             bPar.setUp(); // (I) needs tearDown (see II)
561
562             try {
563                 tool.startMeasuring(benchmark);
564                 benchmarkCounter++;

```

```

565     } catch (IOException e) {
566         logE("Measuring caused IO exception", e);
567         if (mainUI.userWantsToRetry(e)) {
568             continue; // without incrementing i
569         } else {
570             throw new InterruptedException("User wants to abort");
571         }
572     } finally {
573         bPar.tearDown(); // (II) needs setUp (see I)
574     }
575     if (tool.explicitGC() && benchmarkCounter % 25 == 0) {
576         System.gc();
577         Thread.sleep(350);
578     }
579
580     String message = String.format(
581         "%s%s round %d/%d benchmark %d/%d range %d/%d",
582         tool.getClass().getSimpleName(),
583         tool.isWarmupRound() ? " (warmup)" : "",
584         roundNo + 1,
585         roundCount,
586         benchmarkIndex,
587         benchmarkCount,
588         j,
589         dynamicParameters ? BenchmarkParameter.RANGE : 1
590     );
591     mainUI.updateState(ApplicationState.State.MILESTONE, message);
592
593     List<BenchmarkResult> measurements = tool.getMeasurements();
594     for (BenchmarkResult measurement : measurements) {
595         if (!measurement.isEmpty()) {
596             // todo: actual vs. requested size (objects etc.)
597             if (dynamicParameters) {
598                 measurement.put("dynamic_size", "" + i);
599             } else {
600                 measurement.put("dynamic_size", MISSING_VALUE);
601             }
602             measurement.put("id", benchmark.id());
603             measurement.put("class", benchmark.getClass()
604                 .getSimpleName());
605             measurement.putAll(introspected);
606             compiledMetadata.add(measurement);
607         }
608     }
609     // if parameter size can be varied, vary it - else break with
610     // first size
611     if (!dynamicParameters) {
612         break;
613     }
614     if (iterator.hasNext()) {
615         i = iterator.next();
616         j++;
617     } else {

```

```

618         break;
619     }
620 }
621 return compiledMetadata;
622 }
623
624
625 private static BenchmarkResult inspectBenchmark(Benchmark benchmark)
626     throws ClassNotFoundException {
627     BenchmarkResult bdata = new BenchmarkResult();
628     int seqNo = benchmark.sequenceNo();
629     String from = benchmark.from();
630     String to = benchmark.to();
631     bdata.put("no", "" + benchmark.sequenceNo());
632     bdata.put("description", benchmark.description());
633     bdata.put("from", from);
634     bdata.put("to", to);
635     bdata.put("representative", benchmark.representative() ? "1" : "0");
636     bdata.put("nonvirtual", benchmark.isNonvirtual() ? "1" : "0");
637
638     if (seqNo == -1) {
639         bdata.put("custom", "1");
640         return bdata;
641     }
642
643     Class c = Class.forName(
644         String.format(
645             "fi.helsinki.cs.tituomin.nativebenchmark.benchmark" +
646             ".J2CBenchmark%05d",
647             seqNo));
648
649     Method[] methods = c.getDeclaredMethods();
650     for (int i = 0; i < methods.length; i++) {
651
652         Method m = methods[i];
653         int modifiers = m.getModifiers();
654
655         if (Modifier.isNative(modifiers)) {
656             return inspectMethod(m, bdata);
657         }
658     }
659     return bdata;
660 }
661
662 private static BenchmarkResult inspectMethod(Method m, BenchmarkResult
663     bdata) {
664
665     Class[] parameters = m.getParameterTypes();
666     List<Class> parameterList = new ArrayList<Class>(Arrays.asList
667         (parameters));
668     int modifiers = m.getModifiers();
669
670     Map<String, Integer> parameterTypes = new HashMap<String, Integer>();

```

```

671     for (Class param : parameterList) {
672         Integer previousValue = null;
673         String param_typename = param.getSimpleName();
674         previousValue = parameterTypes.get(param_typename);
675         parameterTypes.put(
676             param_typename,
677             (previousValue == null ? 1 : ((int) previousValue) + 1));
678     }
679     for (String typename : parameterTypes.keySet()) {
680         bdata.put("parameter_type_" + typename + "_count", parameterTypes
681             .get(typename) + "");
682     }
683
684     Class returnType = m.getReturnType();
685
686     boolean hasRefTypes = false;
687     parameterList.add(returnType);
688     for (Class cl : parameterList) {
689         if (Object.class.isAssignableFrom(cl)) {
690             hasRefTypes = true;
691             break;
692         }
693     }
694
695     bdata.put("has_reference_types", hasRefTypes ? "1" : "0");
696     bdata.put("parameter_type_count", parameterTypes.keySet().size() + "");
697     bdata.put("parameter_count", parameters.length + "");
698     bdata.put("return_type", returnType.getCanonicalName());
699     bdata.put("native_static", Modifier.isStatic(modifiers) ? "1" : "0");
700     bdata.put("native_private", Modifier.isPrivate(modifiers) ? "1" : "0");
701     bdata.put("native_protected", Modifier.isProtected(modifiers) ? "1" :
702         "0");
703     bdata.put("native_public", Modifier.isPublic(modifiers) ? "1" : "0");
704
705     return bdata;
706 }
707 }

```

BenchmarkSelector.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkInitialiser;
4 //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
5 //import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRunner;
6
7 import android.app.Activity;
8 import android.app.AlertDialog;
9 import android.app.Dialog;

```

```

10 import android.app.DialogFragment;
11 import android.app.Notification;
12 import android.app.NotificationManager;
13 import android.app.PendingIntent;
14 import android.app.TaskStackBuilder;
15 import android.content.Context;
16 import android.content.DialogInterface;
17 import android.content.Intent;
18 import android.content.res.Resources;
19 import android.media.RingtoneManager;
20 import android.net.Uri;
21 import android.os.Bundle;
22 import android.os.Environment;
23 import android.util.Log;
24 import android.view.View;
25 import android.view.WindowManager;
26 import android.widget.AdapterView;
27 import android.widget.AdapterView.OnItemClickListener;
28 import android.widget.ArrayAdapter;
29 import android.widget.Button;
30 import android.widget.Checkable;
31 import android.widget.NumberPicker;
32 import android.widget.Spinner;
33 import android.widget.TextView;
34
35 import java.io.File;
36 import java.io.InputStream;
37 import java.io.OutputStream;
38 import java.util.Map;
39
40 public class BenchmarkSelector extends Activity {
41     /**
42      * Called when the activity is first created.
43      */
44     @Override
45     public void onCreate(Bundle savedInstanceState) {
46         super.onCreate(savedInstanceState);
47         this.requestWindowFeature(getWindow().FEATURE_NO_TITLE);
48         setContentView(R.layout.main);
49
50         this.retry = false;
51
52         this.resultView = (TextView) findViewById(R.id.resultview);
53         this.button = (Button) findViewById(R.id.mybutton);
54         this.repView = (TextView) findViewById(R.id.repetitions);
55         this.numPick = (NumberPicker) findViewById(R.id.picker_num);
56         this.expPick = (NumberPicker) findViewById(R.id.picker_exp);
57
58         NumberPicker.OnValueChangeListener listener = new RepsListener();
59
60         numPick.setMinValue(1);
61         numPick.setMaxValue(9);
62         numPick.setValue(1);

```

```

63     expPick.setMinValue(0);
64     expPick.setMaxValue(9);
65     expPick.setValue(6);
66     numPick.setOnValueChangedListener(listener);
67     expPick.setOnValueChangedListener(listener);
68
69     listener.onValueChange(numPick, 0, 0);
70
71     if (getResources().getString(R.string.app_dirty).equals("1")) {
72         this.resultView.setText(R.string.warning_changed);
73     }
74
75     final Resources resources = getResources();
76     this.runner = BenchmarkRunner.INSTANCE
77         .setAppChecksum(resources.getText(R.string.app_checksum))
78         .setAppRevision(resources.getText(R.string.app_revision))
79         .setCacheDir(getCacheDir());
80
81     File sd = Environment.getExternalStorageDirectory();
82     dataDir = new File(sd, "results");
83     dataDir.mkdir();
84
85     this.controller = new BenchmarkController(this, dataDir);
86
87     this.socketCommunicator = new SocketCommunicator();
88     //this.controller.addListener(socketCommunicator);
89
90     // TODO: configuration is not UI specific.
91     File configFile = new File(
92         Environment.getExternalStorageDirectory(),
93         "nativebenchmark_setup.json"
94     );
95     try {
96         if (!configFile.exists()) {
97             InputStream templateStream = getResources()
98                 .openRawResource(R.raw.setup);
99             OutputStream configFileStream = Utils.makeOutputStream
100                 (configFile, false);
101             Utils.copyStream(templateStream, configFileStream);
102         }
103         this.configuration = ToolConfig.readConfigFile();
104     } catch (Exception e) {
105         String msg = getResources().getString(R.string.config_error);
106         Log.e(TAG, msg, e);
107         displayMessage(ApplicationState.State.INIT_FAIL, msg);
108     }
109
110     if (this.configuration != null) {
111         initSpinner(this.configuration);
112         if (allocationArray == null) {
113             // pre-enlarges the heap
114             // commented out because
115             // space is not reclaimed

```



```

116         // even on gc
117         allocationArray = new byte[1024 * 1024 * 100];
118     }
119 }
120 this.socketCommunicator.startServer(this.controller, this.runner);
121 }
122
123 public void onDestroy() {
124     socketCommunicator.stopServer();
125     super.onDestroy();
126 }
127
128 private void initSpinner(Map<String, ToolConfig> conf) {
129     Spinner spinner = (Spinner) findViewById(R.id.config_spinner);
130     String keys[] = conf.keySet().toArray(new String[1]);
131     ArrayAdapter<CharSequence> adapter = new ArrayAdapter(this, android.R
132         .layout.simple_spinner_item, keys);
133
134     int indexOfDefault = -1;
135     while (++indexOfDefault < keys.length &&
136         !keys[indexOfDefault].equals("default")) ;
137
138     adapter.setDropDownViewResource(android.R.layout
139         .simple_spinner_dropdown_item);
140     spinner.setAdapter(adapter);
141     spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
142         public void onItemSelected(
143             AdapterView<?> parent, View view,
144             int pos, long id) {
145             selectedConfiguration = (String) parent.getItemAtPosition(pos);
146         }
147
148         public void onNothingSelected(AdapterView<?> parent) {
149         }
150     });
151
152     spinner.setSelection(indexOfDefault);
153
154 }
155
156 public void displayMessage(ApplicationState.State state, String message) {
157     displayMessage(state.stringId, message);
158 }
159
160 public void displayMessage(int id) {
161     this.resultView.setText(id);
162 }
163
164 public void displayMessage(String message) {
165     this.resultView.setText(message);
166 }
167
168 public void displayMessage(int id, String message) {

```

```

169         this.resultView.setText(getResources().getString(id) + " " + message);
170     }
171
172     private class StateChanger implements Runnable {
173         public void run() {
174             ApplicationState.DetailedState detailedState = controller
175                 .getState();
176             ApplicationState.State state = detailedState.state;
177             String message = detailedState.message;
178
179             if (message == null) {
180                 displayMessage(state.stringId);
181             } else {
182                 displayMessage(state.stringId, message);
183             }
184             switch (state) {
185                 case MEASURING_STARTED:
186                     getWindow().addFlags(WindowManager.LayoutParams
187                         .FLAG_KEEP_SCREEN_ON);
188                     expPick.setEnabled(false);
189                     numPick.setEnabled(false);
190                     switchButton(button);
191                     state = ApplicationState.State.MILESTONE;
192                     break;
193                 case MILESTONE:
194                     break;
195                 case ERROR:
196                     displayMessage(ApplicationState.State.ERROR, message);
197                     // intended fallthrough
198                 case INTERRUPTED:
199                     // intended fallthrough
200                 case MEASURING_FINISHED:
201                     getWindow().clearFlags(WindowManager.LayoutParams
202                         .FLAG_KEEP_SCREEN_ON);
203                     stateThread.interrupt();
204                     notifyFinished();
205                     // intended fallthrough
206                 case INITIALISED:
207                     resetButton(button);
208                     numPick.setEnabled(true);
209                     expPick.setEnabled(true);
210                     break;
211                 case INIT_FAIL:
212             }
213         }
214     }
215
216     public boolean userWantsToRetry(final Exception e) {
217         runOnUiThread(new Runnable() {
218             public void run() {
219                 DialogFragment dialog = new RetryDialog(e.getMessage());
220                 dialog.show(getFragmentManager(), "foo");
221             }

```

```

222     });
223     boolean waiting = true;
224     while (waiting) {
225         synchronized (this) {
226             try {
227                 this.wait();
228                 waiting = false;
229             } catch (InterruptedException ie) {
230                 waiting = true;
231             }
232         }
233     }
234     return this.retry;
235 }
236
237 private void setRetry(boolean answer) {
238     this.retry = answer;
239     synchronized (this) {
240         this.notify();
241     }
242 }
243
244 private void resetButton(Button button) {
245     button.setText(R.string.start_task);
246     button.setOnClickListener(new View.OnClickListener() {
247         public void onClick(View v) {
248             startMeasuring(v);
249         }
250     });
251 }
252
253 private void switchButton(Button bt) {
254     bt.setText(R.string.end_task);
255     bt.setOnClickListener(
256         new View.OnClickListener() {
257             public void onClick(View v) {
258                 button.setText(R.string.end_task_confirmation);
259                 button.setOnClickListener(
260                     new View.OnClickListener() {
261                         public void onClick(View v) {
262                             displayMessage(ApplicationState.State
263                                 .INTERRUPTING.stringId);
264                             controller.interruptMeasuring();
265                         }
266                     });
267             }
268         });
269 }
270
271
272 private boolean isChecked(int id) {
273     return ((Checkable) findViewById(id)).isChecked();
274 }

```

```

275
276 private String textValue(int id) {
277     return ((TextView) findViewById(id)).getText().toString();
278 }
279
280 public void startMeasuring(View view) {
281     this.runner
282         .setRepetitions(repetitions)
283         .setAllocatingRepetitions(Long.parseLong(textValue(R.id
284             .alloc_reps)))
285         .setBenchmarkSubstring(textValue(R.id.benchmark_substring)
286             .toLowerCase())
287         .setRunAllBenchmarks(isChecked(R.id.checkbox_long))
288         .setRunAtMaxSpeed(isChecked(R.id.checkbox_max))
289         .setBenchmarkSet(isChecked(R.id.run_alloc) ?
290             BenchmarkRunner.BenchmarkSet.ALLOC :
291             BenchmarkRunner.BenchmarkSet.NON_ALLOC);
292
293     stateThread = new Thread(
294         new Runnable() {
295             public void run() {
296                 while (!Thread.currentThread().isInterrupted()) {
297                     runOnUiThread(new StateChanger());
298                     try {
299                         Thread.sleep(5000);
300                     } catch (InterruptedException e) {
301                         break;
302                     }
303                 }
304             }
305         });
306
307     stateThread.start();
308     allocationArray = null;
309     controller.startMeasuring(this.runner, this.configuration.get
310         (selectedConfiguration));
311 }
312
313 private void notifyFinished() {
314     Uri alarmSound = RingtoneManager.getDefaultUri(
315         RingtoneManager.TYPE_NOTIFICATION);
316
317     Notification.Builder mBuilder =
318         new Notification.Builder(this)
319             .setSmallIcon(android.R.drawable.ic_media_play)
320             .setContentTitle(getResources().getText(R.string
321                 .measuring_finished))
322             .setContentText(getResources().getText(R.string
323                 .measuring_finished))
324             .setSound(alarmSound);
325
326     // Creates an explicit intent for an Activity in your app
327     Intent resultIntent = new Intent(this, BenchmarkSelector.class);

```

```

328
329 // The stack builder object will contain an artificial back stack for
330 // the
331 // started Activity.
332 // This ensures that navigating backward from the Activity leads out of
333 // your application to the Home screen.
334 TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
335 // Adds the back stack for the Intent (but not the Intent itself)
336 stackBuilder.addParentStack(BenchmarkSelector.class);
337 // Adds the Intent that starts the Activity to the top of the stack
338 stackBuilder.addNextIntent(resultIntent);
339 PendingIntent resultPendingIntent =
340     stackBuilder.getPendingIntent(
341         0,
342         PendingIntent.FLAG_UPDATE_CURRENT
343     );
344 // mBuilder.setContentIntent(resultPendingIntent);
345 NotificationManager mNotificationManager =
346     (NotificationManager) getSystemService(Context
347         .NOTIFICATION_SERVICE);
348 // mId allows you to update the notification later on.
349 mNotificationManager.notify(1, mBuilder.build());
350 }
351
352 private class RepsListener implements NumberPicker.OnValueChangeListener {
353     public void onValueChange(NumberPicker picker, int oldVal, int newVal) {
354         long exp = BenchmarkSelector.this.expPick.getValue();
355         long value = BenchmarkSelector.this.numPick.getValue();
356         // rounds = BenchmarkSelector.this.roundPick.getValue();
357         while (exp-- > 0) {
358             value *= 10;
359         }
360         repetitions = value;
361         repView.setText("" + repetitions);
362     }
363 }
364
365 private class RetryDialog extends DialogFragment {
366     private String message;
367
368     public RetryDialog(String message) {
369         this.message = message;
370     }
371
372     @Override
373     public Dialog onCreateDialog(Bundle savedInstanceState) {
374         // Use the Builder class for convenient dialog construction
375         AlertDialog.Builder builder = new AlertDialog.Builder(getActivity
376             ());
377         builder.setMessage(getResources().getText(R.string
378             .retry_question) + ":\n" + this.message)
379             .setPositiveButton(R.string.retry_answer_positive, new
380                 DialogInterface.OnClickListener() {

```

```

381         public void onClick(DialogInterface dialog,
382                               int id) {
383             setRetry(true);
384         }
385     })
386     .setNegativeButton(R.string.retry_answer_negative, new
387         DialogInterface.OnClickListener() {
388             public void onClick(DialogInterface dialog,
389                               int id) {
390                 setRetry(false);
391             }
392         });
393     // Create the AlertDialog object and return it
394     return builder.create();
395 }
396 }
397
398 private long repetitions;
399 private boolean retry;
400 private TextView textView, resultView, repView;
401 private NumberPicker numPick, expPick;
402 private Button button;
403 private Thread stateThread;
404 private static byte[] allocationArray;
405 private File dataDir;
406 private BenchmarkController controller;
407 private SocketCommunicator socketCommunicator;
408 private BenchmarkRunner runner;
409
410 private String selectedConfiguration;
411
412 private static final String TAG = "BenchmarkSelector";
413
414 private Map<String, ToolConfig> configuration;
415
416 static {
417     System.loadLibrary("nativebenchmark");
418 }
419
420 }

```

Init.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import java.io.IOException;
4 import java.util.ArrayList;
5 import java.util.List;
6

```

```

7 public class Init {
8
9     private static final String TAG = "NativeBenchmark";
10    public static final int CPUFREQ = 400000;
11    public static final int CPUFREQ_MAX = 1000000;
12
13    public static List<String> initScript(int cpufreq) {
14        List<String> script = new ArrayList<String>();
15        script.add(
16            "echo \"userspace\" > " +
17                "/sys/devices/system/cpu/cpu0/cpufreq" +
18                "/scaling_governor");
19        script.add(
20            "echo \"" + cpufreq + "\" > " +
21                "/sys/devices/system/cpu/cpu0/cpufreq" +
22                "/scaling_setspeed");
23        return script;
24    }
25
26    public static void initEnvironment(boolean maxSpeed) throws IOException {
27        ShellEnvironment.runAsRoot(initScript(maxSpeed ? CPUFREQ_MAX :
28            CPUFREQ));
29    }
30 }
31 }

```

L.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 public class L {
4     // Set L.log to false to statically remove
5     // debugging logging statements from
6     // measuring code.
7     public static final boolean og = false;
8 }

```

LogAccess.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.util.Log;
4
5 import java.io.File;

```

```

6 import java.io.IOException;
7 import java.util.regex.Pattern;
8
9 public class LogAccess {
10
11     private static final String LOGCAT_COMMAND =
12         "logcat -f %s " +
13         "-b main -b system -b radio -b events " +
14         "-v time";
15
16     public static void start(File dir) throws IOException {
17         currentRunId = Utils.getUUID();
18         mark(START, currentRunId);
19         String filename = new File(dir, filename()).getAbsolutePath();
20         String command = String.format(LOGCAT_COMMAND, filename);
21         logcatProcess = Runtime.getRuntime().exec(command);
22     }
23
24     public static void end() {
25         if (logcatProcess != null) {
26             mark(END, currentRunId);
27             logcatProcess.destroy();
28             logcatProcess = null;
29         }
30     }
31
32     private static void mark(String type, String id) {
33         Log.println(LOGLEVEL, TAG, marker(type) + id);
34     }
35
36     private static String marker(String type) {
37         return "[" + type + "] ";
38     }
39
40     public static String filename() {
41         return "log-" + currentRunId + ".txt";
42     }
43     //06-27 23:52:37.348 I/LogAccess( 2378): [End]
44     // 43ba52d0-61b0-47e4-991b-c98a3dd21f9f
45
46     private static Pattern makeMarkerPattern(String type) {
47         return Pattern.compile("[:-. [0-9]]+ I/" + TAG + "\\([ [0-9]+\\): \\[" +
48             type + "\\] " + currentRunId);
49     }
50
51     private static final int LOGLEVEL = Log.INFO;
52     private static final String TAG = "LogAccess";
53     private static final String START = "Start";
54     private static final String END = "End";
55     private static String currentRunId;
56     private static Process logcatProcess;
57 }

```



```
1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
7
8 public class AllocatingBenchmarkLongRunningWrapper extends
9     AllocatingBenchmarkWrapper {
10
11     public AllocatingBenchmarkLongRunningWrapper(Benchmark b, long r) {
12         super(b, r);
13     }
14
15     private static final long MAX_REPS = Long.MAX_VALUE;
16
17     public void begin(MeasuringTool tool) throws InterruptedException,
18         IOException {
19         Benchmark benchmark = getBenchmark();
20         init(benchmark);
21         tool.putMeasurement("repetitions", this.repetitions + "");
22         tool.start(this);
23         tool.finishMeasurement();
24     }
25
26     public void run() {
27         // This method ensures the garbage collector is run
28         // every benchmark.maxrepetitions iteration
29         // but otherwise the measurement is
30         // run for a period long enough for profiling.
31         Benchmark benchmark = getBenchmark();
32         long interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL;
33         long division, remainder;
34         long repetitions = MAX_REPS;
35
36         division = repetitions / interval + 1;
37         remainder = repetitions % interval + 1;
38
39         while (--division != 0) {
40             interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL + 1;
41             while (--interval != 0) {
42                 try {
43                     benchmark.run();
44                     System.gc();
45                     Thread.sleep(GC_PAUSE_MS);
46                 } catch (InterruptedException e) {
47                     setInterrupted();
48                     return;
49                 } catch (Exception e) {
50                     setException(e);
51                 }
41
```

```

51         return;
52     }
53 }
54 if (Thread.currentThread().isInterrupted()) {
55     setInterrupted();
56     return;
57 }
58 }
59
60 while (--remainder != 0) {
61     try {
62         benchmark.run();
63         System.gc();
64         Thread.sleep(GC_PAUSE_MS);
65     } catch (InterruptedException e) {
66         setInterrupted();
67         return;
68     } catch (Exception e) {
69         setException(e);
70         return;
71     }
72 }
73
74 }
75
76 }

```

measuringtool/AllocatingBenchmarkShortRunningWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6
7 public class AllocatingBenchmarkShortRunningWrapper extends
8     AllocatingBenchmarkWrapper {
9
10     public AllocatingBenchmarkShortRunningWrapper(Benchmark b, long r) {
11         super(b, r);
12     }
13
14     private static final long MULTIPLIER = 25;
15
16     public void begin(MeasuringTool tool) throws InterruptedException,
17         IOException {
18         Benchmark benchmark = getBenchmark();
19         init(benchmark);
20         long reps = MULTIPLIER;

```

```

21     reps += 1;
22     while (--reps != 0) {
23         tool.putMeasurement("repetitions", this.repetitions + "");
24         tool.putMeasurement("multiplier", MULTIPLIER + "");
25         try {
26             tool.start(getBenchmark());
27             tool.finishMeasurement();
28             System.gc();
29             Thread.sleep(GC_PAUSE_MS);
30         } catch (InterruptedException e) {
31             setInterrupted();
32             return;
33         } catch (Exception e) {
34             setException(e);
35             return;
36         }
37     }
38 }
39 }

```

measuringtool/AllocatingBenchmarkWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
4
5 public class AllocatingBenchmarkWrapper extends RunningWrapper {
6
7     public AllocatingBenchmarkWrapper(Benchmark b, long repetitions) {
8         super(b);
9         this.repetitions = repetitions;
10    }
11
12    public static final int GC_PAUSE_MS = 400;
13
14    public void init(Benchmark benchmark) {
15        super.init(benchmark);
16        benchmark.setRepetitions(repetitions);
17    }
18
19    protected long repetitions;
20
21 }

```

measuringtool/BasicOption.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.util.Pair;
4
5 public class BasicOption implements MeasuringOption {
6
7     public BasicOption(OptionSpec type) {
8         this(type, null);
9     }
10
11     public BasicOption(OptionSpec type, String value) {
12         this.type = type;
13         this.value = value;
14     }
15
16     // -----
17
18     public OptionSpec id() {
19         return this.type;
20     }
21
22     public void set(String value) {
23         this.value = value;
24     }
25
26     public OptionSpec type() {
27         return this.type;
28     }
29
30     public String value() {
31         return value;
32     }
33
34     public Pair<String, String> toStringPair() {
35         return new Pair<String, String>(this.type.id(), this.value);
36     }
37
38     public String toString() {
39         return this.type() + " " + this.value();
40     }
41
42     // ----
43
44     private OptionSpec type;
45     private String value;
46
47     // ----
48
49 }

```

```
1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.util.Log;
4
5 import java.io.IOException;
6 import java.text.DateFormat;
7 import java.util.ArrayList;
8 import java.util.Date;
9 import java.util.LinkedList;
10 import java.util.List;
11 import java.util.Random;
12
13 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
14 import fi.helsinki.cs.tituomin.nativebenchmark.ShellEnvironment;
15
16
17 public abstract class CommandLineTool extends MeasuringTool {
18
19     // public CommandLineTool(int i, long reps, long allocreps) throws
20     // IOException, InterruptedException {
21     //     super(i, reps, allocreps);
22     // }
23
24     private static final long REPETITIONS = Long.MAX_VALUE;
25
26     public CommandLineTool(int rounds, long repetitions, long allocreps,
27                             boolean warmup, boolean x) throws IOException,
28                             InterruptedException {
29         super(rounds, REPETITIONS, allocreps, false, x);
30         this_filenames = new ArrayList<String>();
31     }
32
33     protected abstract String command();
34
35     protected String formatParameter(MeasuringOption option) {
36         throw new UnsupportedOperationException();
37     }
38
39     protected String formatDefaultParameter(MeasuringOption option) {
40         if (option.type() == OptionSpec.COMMAND_STRING) {
41             return option.value();
42         } else {
43             return formatParameter(option);
44         }
45     }
46
47     public void initCommand() {
48         List<String> commandline = new LinkedList<String>();
49         commandline.addAll(generateCommandLineArguments());
50         this.command = "";
```

```

51     for (String s : commandline) {
52         this.command += s + " ";
53     }
54 }
55
56 protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
57     options = super.specifyAllowedOptions(options);
58     options.add(OptionSpec.COMMAND_STRING);
59     return options;
60 }
61
62 protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
63                                             options) {
64     options.add(new BasicOption(OptionSpec.COMMAND_STRING, command()));
65     return options;
66 }
67
68 protected void init() throws IOException, InterruptedException {
69     super.init();
70     if (!ShellEnvironment.runAsRoot(initScript())) {
71         throw new IOException("Error executing as root.");
72     }
73 }
74
75 protected abstract List<String> initScript();
76
77 public boolean isLongRunning() {
78     return true;
79 }
80
81 public void start(Runnable benchmark)
82     throws InterruptedException, IOException {
83     if (Thread.interrupted()) {
84         throw new InterruptedException();
85     }
86
87     initCommand();
88     BenchmarkRegistry.resetInterruptFlag();
89     Thread benchmarkThread = new Thread(benchmark);
90     Random r = new Random();
91     int delay = r.nextInt(20);
92
93     benchmarkThread.start();
94     Thread.sleep(delay);
95
96     try {
97         ShellEnvironment.run(this.command);
98     } catch (InterruptedException e) {
99         throw e;
100    } finally {
101        benchmarkThread.interrupt();
102        BenchmarkRegistry.interruptNative();
103        benchmarkThread.join();

```

```

104     }
105 }
106
107 public void setFilename(String name, String path) {
108     filenames.add(path + "/" + name);
109     putMeasurement("Filename", name);
110 }
111
112 public List<String> getFilenames() {
113     return filenames;
114 }
115
116
117 public void setUUID(String uuid) {
118     putMeasurement("UUID", uuid);
119 }
120
121 // -----
122
123 public List<String> generateCommandLineArguments() {
124     List<String> result = new LinkedList<String>();
125     for (OptionSpec type : allowedOptions) {
126         MeasuringOption option = options.get(type);
127
128         if (option == null) {
129             Log.v("mt", type + " is null");
130         } else {
131             result.add(formatDefaultParameter(option));
132         }
133     }
134     return result;
135 }
136
137 // -----
138
139 private Date startDate;
140 private Date endDate;
141 private long startTime;
142 private String command;
143
144 private List<String> filenames;
145 private static final DateFormat dateFormat = DateFormat
146     .getDateTimeInstance();
147
148 }

```

measuringtool/JavaSystemNanoResponseTimeRecorder.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 public class JavaSystemNanoResponseTimeRecorder extends ResponseTimeRecorder {
6
7     public JavaSystemNanoResponseTimeRecorder(
8         int i,
9         long reps,
10        long allocreps,
11        boolean warmup,
12        boolean runAllBenchmarks
13    ) throws IOException, InterruptedException {
14        super(i, reps, allocreps, warmup, runAllBenchmarks);
15    }
16
17    public void start(Runnable benchmark)
18        throws InterruptedException, IOException {
19        long endTime, startTime;
20        startTime = System.nanoTime();
21        benchmark.run();
22        endTime = System.nanoTime();
23        String delta = "" + (endTime - startTime);
24        putMeasurement("response_time", delta);
25        putMeasurement("time_unit", "nanoseconds");
26    }
27 }

```

measuringtool/LinuxPerfRecordTool.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.util.LinkedList;
6 import java.util.List;
7
8 import fi.helsinki.cs.tituomin.nativebenchmark.Utills;
9
10 public class LinuxPerfRecordTool extends CommandLineTool {
11
12     // public LinuxPerfRecordTool(int i, long reps, long allocreps) throws
13     // IOException, InterruptedException {
14     //     super(i, reps, allocreps);
15     // }
16
17     public LinuxPerfRecordTool
18         (
19         int rounds,

```



```

20         long repetitions,
21         long allocreps,
22         boolean warmup,
23         boolean runAllBenchmarks) throws
24         IOException, InterruptedException {
25     super(rounds, repetitions, allocreps, warmup, runAllBenchmarks);
26 }
27
28 protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
29     options = super.specifyAllowedOptions(options);
30     options.add(OptionSpec.OUTPUT_FILEPATH);
31     options.add(OptionSpec.MEASURE_LENGTH); // must be last
32     return options;
33 }
34
35 protected List<String> initScript() {
36     List<String> commands = new LinkedList<String>();
37     commands.add("echo \"0\" > /proc/sys/kernel/kptr_restrict");
38     commands.add("echo \"-1\" > /proc/sys/kernel/perf_event_paranoid");
39     return commands;
40 }
41
42 protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
43                                             options) {
44     options = super.defaultOptions(options);
45     String uuid = Utils.getUUID();
46     String filename = generateFilename(uuid);
47     String basePath = getPerfDir().getPath() + "/";
48     setFilename(filename, basePath);
49     setUUID(uuid);
50     options.add(new BasicOption(OptionSpec.OUTPUT_FILEPATH, basePath +
51                               filename));
52     return options;
53 }
54
55 protected void init() throws IOException, InterruptedException {
56     super.init();
57     getPerfDir().mkdir();
58 }
59
60 private File getPerfDir() {
61     return new File(getDataDir(), "perf");
62 }
63
64 protected String command() {
65     return "perf record -a -g";
66 }
67
68 private String generateFilename(String uuid) {
69     return "perf-" + uuid + ".data";
70 }
71
72 public String formatParameter(MeasuringOption option) {

```

```

73         if (option.type() == OptionSpec.OUTPUT_FILEPATH) {
74             return "--output=" + option.value();
75         } else if (option.type() == OptionSpec.MEASURE_LENGTH) {
76             return "sleep " + option.value();
77         }
78         return super.formatParameter(option);
79     }
80 }

```

measuringtool/MeasuringOption.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.util.Pair;
4
5 public interface MeasuringOption {
6
7     public void set(String value);
8
9     public OptionSpec id();
10
11     public String value();
12
13     public OptionSpec type();
14
15     public Pair<String, String> toStringPair();
16 }

```

measuringtool/MeasuringTool.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3
4 import android.util.Log;
5 import android.util.Pair;
6
7 import java.io.File;
8 import java.io.IOException;
9 import java.text.SimpleDateFormat;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import java.util.HashMap;
13 import java.util.HashSet;
14 import java.util.LinkedList;

```

```

15 import java.util.List;
16 import java.util.Locale;
17 import java.util.Map;
18 import java.util.Set;
19
20 import fi.helsinki.cs.tituomin.nativebenchmark.ApplicationState;
21 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
22 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
23 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkResult;
24
25 public abstract class MeasuringTool implements Runnable {
26
27     public MeasuringTool
28         (
29             int rounds,
30             long repetitions,
31             long allocRepetitions,
32             boolean warmup,
33             boolean runAllBenchmarks
34         ) throws
35             IOException, InterruptedException {
36         clearMeasurements();
37         specifyOptions();
38         this.rounds = rounds;
39         this.repetitions = repetitions;
40         this.allocRepetitions = allocRepetitions;
41         this.warmup = warmup;
42         this.explicitGC = !warmup;
43         this.runAllBenchmarks = runAllBenchmarks;
44         init();
45     }
46
47     public static synchronized void userInterrupt() {
48         userInterrupted = true;
49         BenchmarkRegistry.interruptNative();
50     }
51
52     public static synchronized boolean userInterrupted() {
53         if (userInterrupted == true) {
54             userInterrupted = false;
55             return true;
56         }
57         return false;
58     }
59
60     protected void init() throws IOException, InterruptedException {
61     }
62
63     protected abstract void start(Runnable benchmark)
64         throws InterruptedException, IOException;
65
66     public boolean isLongRunning() {
67         return false;

```

```

68     }
69
70     public RunningWrapper wrap(Benchmark benchmark) {
71         if (!benchmark.isAllocating()) {
72             // Default running algorithm
73             return new RunningWrapper(benchmark);
74         } else {
75             // the benchmark does allocations, we have
76             // to limit the amount of loops -> compensate
77             // by repeating the loop many times
78             if (isLongRunning()) {
79                 return new AllocatingBenchmarkLongRunningWrapper(benchmark,
80                     allocRepetitions);
81             } else {
82                 return new AllocatingBenchmarkShortRunningWrapper(benchmark,
83                     allocRepetitions);
84             }
85         }
86     }
87
88     public void startMeasuring(Benchmark benchmark) throws
89         InterruptedException, IOException, RunnerException {
90         String benchmarkName = benchmark.getClass().getSimpleName();
91         clearMeasurements();
92         setDefaultOptions();
93         benchmark.setRepetitions(this.repetitions);
94         RunningWrapper wrapper = wrap(benchmark);
95         Date start = null, end = null;
96
97         start = new Date();
98         wrapper.begin(this);
99         end = new Date();
100
101         putMeasurement("start", DATEFORMAT.format(start));
102         putMeasurement("end", DATEFORMAT.format(end));
103
104         if (wrapper.wasInterrupted() && userInterrupted()) {
105             throw new InterruptedException("Interrupted by user");
106         }
107         if (wrapper.exceptionWasThrown()) {
108             throw new RunnerException(wrapper.getException());
109         }
110     }
111
112     public class RunnerException extends Exception {
113         public RunnerException(Throwable t) {
114             super(t);
115         }
116     }
117
118
119     public void run() {
120         try {

```

```

121         start(benchmark);
122     } catch (InterruptedException e) {
123         Log.e("BenchmarkRunner", "Measuring was interrupted ", e);
124     } catch (IOException e) {
125         Log.e("BenchmarkRunner", "IOException", e);
126     }
127 }
128
129 public boolean explicitGC() {
130     return this.explicitGC;
131 }
132
133 public boolean runAllBenchmarks() {
134     return this.runAllBenchmarks;
135 }
136
137 public void setExplicitGC(boolean e) {
138     this.explicitGC = e;
139 }
140
141 protected abstract List<MeasuringOption>
142 defaultOptions(List<MeasuringOption> container);
143
144 protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec>
145                                                     container) {
146     return container;
147 }
148
149 protected void specifyOptions() {
150     this.allowedOptions = specifyAllowedOptions(
151         new LinkedList<OptionSpec>());
152
153     if (this.requiredOptions == null) {
154         this.requiredOptions = new HashSet<OptionSpec>();
155     }
156     for (OptionSpec o : this.allowedOptions) {
157         if (o.required()) {
158             this.requiredOptions.add(o);
159         }
160     }
161 }
162
163 protected void setDefaultOptions() {
164     for (MeasuringOption op : defaultOptions(
165         new LinkedList<MeasuringOption>())) {
166         setOption(op);
167     }
168     ;
169 }
170
171 // -----
172
173 public MeasuringTool set(String id, String value) {

```

```

174         setOption(new BasicOption(OptionSpec.byId(id), value));
175         return this;
176     }
177
178     public MeasuringTool set(OptionSpec spec, String value) {
179         // todo: not typesafe (assumes basicoption)
180         setOption(new BasicOption(spec, value));
181         return this;
182     }
183
184     public String getOption(OptionSpec id) {
185         return this.options.get(id).value();
186     }
187
188     public void setDescription(String d) {
189         this.description = d;
190     }
191
192     public void setOption(MeasuringOption option) {
193         if (this.allowedOptions == null) {
194             specifyOptions();
195         }
196         if (!allowedOptions.contains(option.type())) {
197             throw new UnsupportedOperationException();
198         } else {
199             this.options =
200                 options != null ?
201                     options :
202                     new HashMap<OptionSpec, MeasuringOption>();
203
204             this.options.put(option.type(), option);
205         }
206     }
207
208     public void setBenchmark(Benchmark b) {
209         benchmark = b;
210     }
211
212     public boolean ignore() {
213         return false;
214     }
215
216     private boolean hasRequiredOptions() {
217         return options.keySet().containsAll(requiredOptions);
218     }
219
220     protected List<OptionSpec> allowedOptions;
221     protected Set<OptionSpec> requiredOptions;
222     // currently support one value per option, change?
223     protected Map<OptionSpec, MeasuringOption> options;
224
225     private List<ApplicationState> observers;
226

```

```

227 private BenchmarkResult currentMeasurement;
228 private List<BenchmarkResult> measurements;
229
230 protected long repetitions;
231
232 protected void putMeasurement(String key, String value) {
233     currentMeasurement.put(key, value);
234 }
235
236 protected void finishMeasurement() {
237     currentMeasurement = new BenchmarkResult();
238     measurements.add(currentMeasurement);
239 }
240
241 public List<BenchmarkResult> getMeasurements() {
242     for (BenchmarkResult measurement : measurements) {
243         if (this.options != null) {
244             for (MeasuringOption op : options.values()) {
245                 measurement.put(
246                     op.toStringPair().first,
247                     op.toStringPair().second);
248             }
249         }
250     }
251     return this.measurements;
252 }
253
254 public void clearMeasurements() {
255     this.currentMeasurement = new BenchmarkResult();
256     this.measurements = new LinkedList<BenchmarkResult>();
257     measurements.add(currentMeasurement);
258 }
259
260 private static final List<String> emptyList = new ArrayList<String>();
261
262 public List<String> getFilenames() {
263     return emptyList;
264 }
265
266 public int getRounds() {
267     return rounds;
268 }
269
270 public static void setDataDir(File dir) {
271     dataDir = dir;
272 }
273
274 public static File getDataDir() {
275     return dataDir;
276 }
277
278 public void setFilter(String substring) {
279     filterSubstring = substring;

```

```

280     }
281
282     public String getFilter() {
283         return filterSubstring;
284     }
285
286     public List<Pair<String, String>> configuration() {
287         List<Pair<String, String>> pairs = new ArrayList<Pair<String,
288             String>>();
289         pairs.add(new Pair<String, String>("tool", this.getClass()
290             .getSimpleName()));
291         pairs.add(new Pair<String, String>("repetitions", "" + this
292             .repetitions));
293         pairs.add(new Pair<String, String>("description", this.description));
294         pairs.add(new Pair<String, String>("warmup", "" + this.warmup));
295         if (options != null) {
296             for (MeasuringOption opt : options.values()) {
297                 pairs.add(new Pair<String, String>(opt.type().id(), opt.value
298                     ()));
299             }
300         }
301         return pairs;
302     }
303
304     public boolean isWarmupRound() {
305         return warmup;
306     }
307
308     private static File dataDir;
309
310     private Benchmark benchmark;
311     private int rounds;
312     private long allocRepetitions;
313     private String description;
314     private String filterSubstring;
315     protected boolean warmup;
316     private boolean explicitGC;
317     private boolean runAllBenchmarks;
318     private static boolean userInterrupted = false;
319     private final static String TAG = "MeasuringTool";
320
321     public static class UnsupportedOperationException extends RuntimeException {
322     }
323
324     private static SimpleDateFormat DATEFORMAT = new SimpleDateFormat("MM-dd " +
325         "HH:mm:ss.SSS", Locale.US);
326
327 }

```

measuringtool/MockCommandlineTool.java


```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 public class MockCommandlineTool extends CommandlineTool {
7
8     public MockCommandlineTool(int i, long reps) throws IOException,
9         InterruptedException {
10         super(i, reps, reps, false, false);
11     }
12
13     protected List<String> initScript() {
14         return null;
15     }
16
17     public boolean ignore() {
18         return true;
19     }
20
21     protected String command() {
22         return "cat /dev/null";
23     }
24
25 }

```

measuringtool/OptionSpec.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public enum OptionSpec {
7
8     COMMAND_STRING("Command run", "command_string", true),
9     OUTPUT_FILEPATH("Output path", "output_filepath", true),
10    MEASURE_LENGTH("Measuring time (sec)", "measure_length", true),
11    VARIABLE("Variable parameter in benchmark", "variable", false),
12    CPUFREQ("Fixed CPU frequency", "cpu_freq", true);
13
14    OptionSpec(String name, String id, boolean required) {
15        this.name = name;
16        this.id = id;
17        this.required = required;
18        put(id, this);
19    }
20
21    public String id() {

```

```

22         return id;
23     }
24
25     public boolean required() {
26         return required;
27     }
28
29     public static OptionSpec byId(String id) {
30         return specs.get(id);
31     }
32
33     private static void put(String id, OptionSpec value) {
34         getSpecs().put(id, value);
35     }
36
37     public static Map<String, OptionSpec> getSpecs() {
38         if (specs == null) {
39             specs = new HashMap<String, OptionSpec>();
40         }
41         return specs;
42     }
43
44     private String name;
45     private String id;
46     private boolean required;
47
48     private static Map<String, OptionSpec> specs;
49
50 }

```

measuringtool/PlainRunner.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4 import java.util.List;
5
6 public class PlainRunner extends MeasuringTool {
7
8     public PlainRunner(int i, long reps, long allocreps) throws IOException,
9         InterruptedException {
10         super(i, reps, allocreps, true, false);
11     }
12
13     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
14         options) {
15         return options;
16     }
17

```

```

18     protected List<OptionSpec> specifyAllowedOptions(List<OptionSpec> options) {
19         options = super.specifyAllowedOptions(options);
20         options.add(OptionSpec.CPUFREQ);
21         return options;
22     }
23
24     public boolean explicitGC() {
25         return false;
26     }
27
28     public boolean ignore() {
29         return true;
30     }
31
32     public long repetitions() {
33         return 10000;
34     }
35
36     public void start(Runnable benchmark)
37         throws InterruptedException, IOException {
38         benchmark.run();
39     }
40 }

```

measuringtool/ResponseTimeRecorder.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import android.os.SystemClock;
4
5 import java.io.IOException;
6 import java.util.List;
7
8 public class ResponseTimeRecorder extends MeasuringTool {
9
10     public ResponseTimeRecorder(
11         int rounds,
12         long reps,
13         long allocreps,
14         boolean warmup,
15         boolean x)
16         throws IOException, InterruptedException {
17         super(rounds, reps, allocreps, warmup, x);
18     }
19
20     protected List<MeasuringOption> defaultOptions(List<MeasuringOption>
21                                             options) {
22         // no options needed, time is returned as is (not in extra file)
23         return options;

```

```

24     }
25
26     public boolean ignore() {
27         return warmup;
28     }
29
30     public void start(Runnable benchmark)
31         throws InterruptedException, IOException {
32         long endTime, startTime;
33         startTime = SystemClock.uptimeMillis();
34         benchmark.run();
35         endTime = SystemClock.uptimeMillis();
36         String delta = "" + (endTime - startTime);
37         putMeasurement("response_time", delta);
38         putMeasurement("time_unit", "milliseconds");
39     }
40
41 }

```

measuringtool/RunningWrapper.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark.measuringtool;
2
3 import java.io.IOException;
4
5 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
6
7 public class RunningWrapper implements Runnable {
8     private Benchmark benchmark;
9     private Exception exceptionThrown;
10    private boolean interrupted;
11    private long repetitions;
12
13    public RunningWrapper(Benchmark b) {
14        benchmark = b;
15    }
16
17    protected void setException(Exception e) {
18        exceptionThrown = e;
19    }
20
21    protected void setInterrupted() {
22        interrupted = true;
23    }
24
25    public boolean exceptionWasThrown() {
26        return exceptionThrown != null;
27    }
28

```

```

29     public void setRepetitions(long r) {
30         repetitions = r;
31     }
32
33     public Benchmark getBenchmark() {
34         return benchmark;
35     }
36
37     public long getRepetitions() {
38         return repetitions;
39     }
40
41     public boolean wasInterrupted() {
42         return interrupted;
43     }
44
45     public Exception getException() {
46         return exceptionThrown;
47     }
48
49     public void init(Benchmark benchmark) {
50         interrupted = false;
51         exceptionThrown = null;
52     }
53
54     public void run() {
55         benchmark.run();
56     }
57
58     public void begin(MeasuringTool tool) throws InterruptedException,
59         IOException {
60         init(benchmark);
61         tool.start(this);
62         if (Thread.currentThread().isInterrupted()) {
63             setInterrupted();
64         }
65     }
66
67 }

```

MockObject.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 public class MockObject {
4
5     public boolean jbooleanField;
6     public byte jbyteField;
7     public char jcharField;

```

```

8      public double jdoubleField;
9      public float jfloatField;
10     public int jintField;
11     public long jlongField;
12     public short jshortField;
13     public Object jobjectField;
14
15     public static boolean jbooleanStaticField;
16     public static byte jbyteStaticField;
17     public static char jcharStaticField;
18     public static double jdoubleStaticField;
19     public static float jfloatStaticField;
20     public static int jintStaticField;
21     public static long jlongStaticField;
22     public static short jshortStaticField;
23     public static Object jobjectStaticField;
24
25     public MockObject() {
26         jbooleanField = true;
27         jbyteField = 1;
28         jcharField = 2;
29         jdoubleField = 1.2;
30         jfloatField = 3.1f;
31         jintField = 3;
32         jlongField = 4;
33         jshortField = 5;
34         jobjectField = this;
35
36         jbooleanStaticField = true;
37         jbyteStaticField = 1;
38         jcharStaticField = 2;
39         jdoubleStaticField = 1.2;
40         jfloatStaticField = 3.1f;
41         jintStaticField = 3;
42         jlongStaticField = 4;
43         jshortStaticField = 5;
44         jobjectStaticField = this;
45     }
46 }

```

ShellEnvironment.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.util.Log;
4
5 import java.io.BufferedReader;
6 import java.io.DataOutputStream;
7 import java.io.IOException;

```

```

8 import java.io.InputStream;
9 import java.io.InputStreamReader;
10 import java.util.List;
11
12 public class ShellEnvironment {
13
14     // thanks http://muzikant-android.blogspot
15     // .fi/2011/02/how-to-get-root-access-and-execute.html
16     public static boolean runAsRoot(List<String> commands) throws IOException {
17         if (commands == null) {
18             return true;
19         }
20         boolean retval = false;
21         DataOutputStream os = null;
22         Process suProcess = null;
23         try {
24             if (null != commands && commands.size() > 0) {
25                 suProcess = Runtime.getRuntime().exec("su");
26
27                 os = new DataOutputStream(suProcess.getOutputStream());
28
29                 // Execute commands that require root access
30                 for (String currCommand : commands) {
31                     os.writeBytes(currCommand + "\n");
32                     os.flush();
33                 }
34
35                 os.writeBytes("exit\n");
36                 os.flush();
37
38                 try {
39                     int suProcessRetval = suProcess.waitFor();
40                     if (255 != suProcessRetval) {
41                         // Root access granted
42                         retval = true;
43                     } else {
44                         // Root access denied
45                         retval = false;
46                     }
47                 } catch (Exception ex) {
48                     Log.e("ROOT", "Error executing root action", ex);
49                 }
50             }
51         } catch (IOException ex) {
52             Log.w("ROOT", "Can't get root access", ex);
53         } catch (SecurityException ex) {
54             Log.w("ROOT", "Can't get root access", ex);
55         } catch (Exception ex) {
56             Log.w("ROOT", "Error executing internal operation", ex);
57         } finally {
58             if (suProcess != null) {
59                 suProcess.destroy();
60             }
61         }
62     }
63 }

```

```

61     }
62     if (os != null) {
63         os.close();
64     }
65
66     return retval;
67 }
68
69 public static void run(String command)
70     throws IOException, InterruptedException {
71
72     Process process = null;
73     InputStream err = null;
74     try {
75         process = Runtime.getRuntime().exec(command);
76         err = process.getErrorStream();
77         process.waitFor();
78
79         if (process.exitValue() != 0) {
80             String line;
81             BufferedReader br = new BufferedReader(new InputStreamReader
82                 (err));
83             StringBuilder sb = new StringBuilder("Command failed.\n");
84             while ((line = br.readLine()) != null) {
85                 Log.e("tm", line);
86                 sb.append(line);
87                 sb.append("\n");
88             }
89             process.destroy();
90             br.close();
91             throw new IOException(sb.toString());
92         }
93     } catch (IOException e) {
94         throw e;
95     } catch (InterruptedException e) {
96         throw e;
97     } finally {
98         if (err != null) err.close();
99         if (process != null) process.destroy();
100     }
101 }
102
103 }

```

SocketCommunicator.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3

```



```

4 import android.util.Log;
5
6 import java.io.IOException;
7 import java.io.InputStream;
8 import java.io.OutputStream;
9 import java.io.PrintWriter;
10 import java.net.InetSocketAddress;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.net.SocketTimeoutException;
14 import java.util.Map;
15
16 public class SocketCommunicator implements ApplicationStateListener {
17     /**
18      * Thread to initialize Socket connection
19      */
20     private final Runnable InitializeConnection = new Thread() {
21         @Override
22         public void run() {
23             // initialize server socket
24             while (!Thread.currentThread().isInterrupted()) {
25                 try {
26                     server = new ServerSocket();
27                     //server.setSoTimeout(TIMEOUT * 1000);
28                     server.setReuseAddress(true);
29                     server.bind(new InetSocketAddress(38300));
30
31                     //attempt to accept a connection
32                     client = server.accept();
33
34                     SocketCommunicator.this.out = client.getOutputStream();
35                     SocketCommunicator.this.printWriter = new PrintWriter
36                         (out, true);
37                     SocketCommunicator.nis = client.getInputStream();
38                     try {
39                         SocketCommunicator.this.output(helpMessage);
40
41                         byte[] bytes = new byte[1024];
42                         int numRead = 0;
43                         while (numRead >= 0) {
44                             SocketCommunicator.this.output("[Awaiting input]");
45                             numRead = SocketCommunicator.nis.read(bytes, 0,
46                                 1024);
47                             if (numRead < 1) {
48                                 executeCommand("quit");
49                                 return;
50                             }
51                             receivedCommand = new String(bytes, 0, numRead)
52                                 .trim();
53                             executeCommand(receivedCommand);
54                         }
55                     } catch (IOException ioException) {
56                         Log.e(SocketCommunicator.TAG, "" + ioException);

```

```

57         }
58     } catch (SocketTimeoutException e) {
59         receivedCommand = "Connection has timed out! Please try " +
60             "again";
61         Log.v(TAG, receivedCommand);
62     } catch (IOException e) {
63         Log.e(SocketCommunicator.TAG, "" + e);
64     }
65
66     if (client != null) {
67         receivedCommand = "Connection was successful!";
68         try {
69             server.close();
70         } catch (IOException e) {
71             Log.e(TAG, "Error closing server connection.");
72         } finally {
73             server = null;
74         }
75         Log.v(TAG, receivedCommand);
76     }
77 }
78
79 };
80
81 private class StateChanger implements Runnable {
82     public void run() {
83     }
84 }
85
86 private void executeStart(String command) {
87     String[] split = command.split(":");
88     if (split.length < 2) {
89         Log.e(TAG, "No configuration key provided.");
90         return;
91     }
92     String configKey = split[1];
93     Map<String, ToolConfig> configurations = null;
94     ToolConfig config = null;
95     try {
96         configurations = ToolConfig.readConfigFile();
97         config = configurations.get(configKey);
98     } catch (Exception e) {
99         Log.e(TAG, "Error reading configuration file.", e);
100     }
101     if (config == null) {
102         ApplicationState.DetailedState ds = new ApplicationState
103             .DetailedState(controller);
104         ds.state = ApplicationState.State.ERROR;
105         ds.message = "Could not find configuration for " + configKey;
106         stateUpdated(ds);
107         return;
108     }
109     stateThread = new Thread(

```

```

110         new Runnable() {
111             public void run() {
112                 while (!Thread.currentThread().isInterrupted()) {
113                     ApplicationState.DetailedState detailedState =
114                         controller.getState();
115                     ApplicationState.State state = detailedState.state;
116                     stateUpdated(detailedState);
117                     if (state == ApplicationState.State
118                         .MEASURING_FINISHED ||
119                         state == ApplicationState.State.ERROR) {
120                         return;
121                     }
122                     try {
123                         Thread.sleep(10000);
124                     } catch (InterruptedException e) {
125                         break;
126                     }
127                 }
128             }
129         }
130     );
131
132     stateThread.start();
133     this.controller.startMeasuring(this.runner, config);
134 }
135
136 private void output(String message) {
137     synchronized (this) {
138         this.printWriter.println(message);
139     }
140 }
141
142 public void executeCommand(String command) {
143     boolean executed = false;
144     command = command.trim();
145     if (command.length() == 0) {
146         return;
147     } else if (command.startsWith("start")) {
148         this.executeStart(command);
149         executed = true;
150     } else if (command.startsWith("end")) {
151         this.controller.interruptMeasuring();
152         executed = true;
153     } else if (command.startsWith("quit")) {
154         this.restartServer();
155         executed = true;
156     } else {
157         this.output(String.format("Unkown command %s", command));
158         Log.v(TAG, "" + command + " == unknown command.");
159     }
160     if (!executed) {
161         return;
162     }

```

```

163         this.output(String.format(
164             "[Executed %s]", command));
165     }
166
167     public void startServer(
168         BenchmarkController controller,
169         BenchmarkRunner runner)
170
171     {
172         this.controller = controller;
173         //this.controller.addListener(this, ApplicationState.State.MILESTONE);
174         this.runner = runner;
175         //initialize server socket in a new separate thread
176         this.serverThread = new Thread(InitializeConnection);
177         this.serverThread.start();
178         String msg = "Attempting to connect";
179         Log.v(TAG, msg);
180     }
181
182     public boolean stopServer() {
183         try {
184             // TODO
185             // Close the opened resources on activity destroy
186             this.output("Stopping socket server.");
187             serverThread.interrupt();
188             if (SocketCommunicator.nis != null) {
189                 SocketCommunicator.nis.close();
190             }
191             if (this.out != null) {
192                 this.out.close();
193             }
194             if (server != null) {
195                 server.close();
196             }
197             return true;
198         } catch (IOException ec) {
199             Log.e(SocketCommunicator.TAG, "Cannot close server socket" + ec);
200             return false;
201         }
202     }
203
204     public void restartServer() {
205         if (this.stopServer()) {
206             this.startServer(this.controller, this.runner);
207         }
208     }
209
210     public void stateUpdated(ApplicationState.DetailedState state) {
211         this.output(state.toString());
212     }
213
214     public static final String TAG = "SocketCommunicator";
215     public static final int TIMEOUT = 10;

```

```

216     private ServerSocket server = null;
217     private Socket client = null;
218     private OutputStream out;
219     private PrintWriter printWriter;
220     private String receivedCommand = null;
221     public static InputStream nis = null;
222     private BenchmarkController controller;
223     private Map<String, ToolConfig> configurations;
224     private BenchmarkRunner runner;
225     private Thread serverThread;
226     private Thread stateThread;
227
228     private final String helpMessage = "\n" +
229         "Measuring application ready.\n" +
230         "Available commands:\n" +
231         "  start :CONFIG_KEY\n" +
232         "    Starts measuring with the configuration\n" +
233         "    loaded from nativebenchmark_setup.json file\n" +
234         "    under the top level key CONFIG_KEY.\n" +
235         "  end\n" +
236         "    Interrupts measuring.\n";
237
238 }

```

ToolConfig.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import android.os.Environment;
4 import android.util.Log;
5
6 import org.json.JSONArray;
7 import org.json.JSONException;
8 import org.json.JSONObject;
9
10 import java.io.File;
11 import java.io.IOException;
12 import java.lang.reflect.Constructor;
13 import java.lang.reflect.InvocationTargetException;
14 import java.util.HashMap;
15 import java.util.Iterator;
16 import java.util.Map;
17 import java.util.NoSuchElementException;
18
19 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.MeasuringTool;
20
21 public class ToolConfig implements Iterable<MeasuringTool> {
22
23     public static Map<String, ToolConfig> readConfigurations(String jsonConfig)

```

```

24         throws JSONException {
25     JSONObject cfgObject = new JSONObject(jsonConfig);
26     Map<String, ToolConfig> result = new HashMap<String, ToolConfig>();
27     Iterator i = cfgObject.keys();
28     while (i.hasNext()) {
29         String key = (String) i.next();
30         result.put(key, new ToolConfig(cfgObject.getJSONObject(key)));
31     }
32     return result;
33 }
34
35 public static Map<String, ToolConfig> readConfigFile() throws
36     IOException, JSONException {
37     String jsonContents = null;
38     File configFile = new File(
39         Environment.getExternalStorageDirectory(),
40         "nativebenchmark_setup.json"
41     );
42     jsonContents = Utils.readFileToString(configFile);
43     return ToolConfig.readConfigurations(jsonContents);
44 }
45
46 public ToolConfig(JSONObject job) {
47     this.globalOptions = job;
48     this.defaultAllocRepetitions = 400;
49 }
50
51 public String toString() {
52     return this.globalOptions.toString();
53 }
54
55 public Iterator<MeasuringTool> iterator() {
56     try {
57         return new ToolIterator();
58     } catch (JSONException e) {
59         Log.e("ToolConfig", "Error reading json config", e);
60     }
61     return null;
62 }
63
64 private class ToolIterator implements Iterator<MeasuringTool> {
65     private JSONArray toolArray;
66     private int currentToolIndex;
67
68     public ToolIterator() throws JSONException {
69         currentToolIndex = -1;
70         toolArray = globalOptions.getJSONArray("tools");
71     }
72
73     public boolean hasNext() {
74         return currentToolIndex + 1 < toolArray.length();
75     }
76 }

```

```

77     public MeasuringTool next() {
78         MeasuringTool tool = null;
79         try {
80             tool = createTool(toolArray.getJSONObject(++currentToolIndex));
81             if (tool == null) {
82                 throw new NoSuchElementException();
83             }
84         } catch (JSONException e) {
85             Log.e("ToolConfig", "Error reading json config", e);
86         }
87         return tool;
88     }
89
90     public void remove() {
91         throw new UnsupportedOperationException();
92     }
93 }
94
95 private MeasuringTool createTool(JSONObject toolOptions) {
96
97     MeasuringTool tool = null;
98     try {
99         long repetitions = toolOptions.optLong(
100             "repetitions", globalOptions.optLong(
101                 toolOptions.optString("repetitions"), this
102                     .defaultRepetitions));
103
104         // todo
105         int defaultRounds = 1;
106         // todo
107
108         int rounds = toolOptions.optInt(
109             "rounds", globalOptions.optInt(
110                 toolOptions.optString("rounds"), defaultRounds));
111
112         long allocRepetitions = toolOptions.optLong(
113             "alloc_repetitions", globalOptions.optLong(
114                 toolOptions.optString("alloc_repetitions"),
115                 this.defaultAllocRepetitions));
116
117         boolean warmup = toolOptions.optBoolean("warmup", false);
118
119         boolean runAllBenchmarks = toolOptions.optBoolean(
120             "run_all", this.defaultRunAllBenchmarks);
121
122         Class<?> _class = Class.forName(TOOL_PACKAGE + "." + toolOptions
123             .getString("class"));
124
125         Constructor<?> ctor = _class.getConstructor(
126             Integer.TYPE, Long.TYPE, Long.TYPE, Boolean.TYPE, Boolean
127                 .TYPE);
128
129         Log.v("ToolConfig", "Tool instantiation " + rounds + " " +

```

```

130         repetitions + " " + warmup);
131
132     try {
133         tool = (MeasuringTool) ctor.newInstance(
134             rounds, repetitions, allocRepetitions,
135             warmup, runAllBenchmarks);
136     } catch (InvocationTargetException e) {
137         Log.e("ToolConfig", "Constructor exception", e.getCause());
138     }
139     tool.setDescription(toolOptions.optString("description", ""));
140     tool.setFilter(toolOptions.optString("filter", ""));
141     tool.setExplicitGC(toolOptions.optBoolean("gc", !warmup));
142
143     JSONObject options = toolOptions.optJSONObject("options");
144     if (options != null) {
145         Iterator keys = options.keys();
146         while (keys.hasNext()) {
147             String key = (String) keys.next();
148             tool.set(key, options.getString(key));
149         }
150     }
151
152     } catch (Exception e) {
153         Log.e("ToolConfig", "Error instantiating tool", e);
154         return null;
155     }
156     return tool;
157 }
158
159 public ToolConfig setDefaultRepetitions(long r) {
160     defaultRepetitions = r;
161     return this;
162 }
163
164 public ToolConfig setDefaultAllocRepetitions(long r) {
165     defaultAllocRepetitions = r;
166     return this;
167 }
168
169 public ToolConfig setDefaultRunAllBenchmarks(boolean r) {
170     defaultRunAllBenchmarks = r;
171     return this;
172 }
173
174 public BenchmarkRunner.BenchmarkSet getBenchmarkSet() {
175     String key = globalOptions.optString("benchmark_set", "non_alloc");
176     if (key.equals("alloc")) {
177         return BenchmarkRunner.BenchmarkSet.ALLOC;
178     }
179     return BenchmarkRunner.BenchmarkSet.NON_ALLOC;
180 }
181
182 private long defaultRepetitions;

```



```

183     private long defaultAllocRepetitions;
184     private boolean defaultRunAllBenchmarks;
185
186     private JSONObject globalOptions;
187     private static final String TOOL_PACKAGE = "fi.helsinki.cs.tituomin" +
188         ".nativebenchmark.measuringtool";
189     private static final String TAG = "nativebenchmark.ToolConfig";
190
191 }

```

Utils.java

```

1 package fi.helsinki.cs.tituomin.nativebenchmark;
2
3 import java.io.BufferedInputStream;
4 import java.io.BufferedOutputStream;
5 import java.io.File;
6 import java.io.FileInputStream;
7 import java.io.FileNotFoundException;
8 import java.io.FileOutputStream;
9 import java.io.IOException;
10 import java.io.InputStream;
11 import java.io.OutputStream;
12 import java.io.PrintWriter;
13 import java.nio.MappedByteBuffer;
14 import java.nio.channels.FileChannel;
15 import java.nio.charset.Charset;
16 import java.util.UUID;
17
18
19 public class Utils {
20
21     public static String getUUID() {
22         return UUID.randomUUID().toString();
23     }
24
25     public static String humanTime(long millis) {
26         String time;
27         long seconds = millis / 1000;
28         long minutes = seconds / 60;
29         long hours = minutes / 60;
30         long seconds_total = seconds;
31         seconds %= 60;
32         minutes %= 60;
33         return (hours + "h " +
34             minutes + "m " +
35             seconds + "s " +
36             "(" + seconds_total + " s tot.)");
37     }

```

```

38
39 public static String colFmt(String label, String value) {
40     return String.format("%20s: %s", label, value);
41 }
42
43 public static void colPr(PrintWriter p, String label, Object value) {
44     p.println(colFmt(label, value.toString()));
45 }
46
47 public static void copyStream(InputStream in, OutputStream out) throws
48     IOException {
49     int count;
50     while ((count = in.read(buffer, 0, BUFFERSIZE)) != -1) {
51         out.write(buffer, 0, count);
52     }
53     out.flush();
54 }
55
56 public static PrintWriter
57 makeWriter(File dir, String filename, boolean append)
58     throws FileNotFoundException {
59     return new PrintWriter(makeOutputStream(dir, filename, append));
60 }
61
62 public static PrintWriter
63 makeWriter(File file, boolean append)
64     throws FileNotFoundException {
65     return new PrintWriter(makeOutputStream(file, append));
66 }
67
68 public static OutputStream
69 makeOutputStream(File dir, String filename, boolean append)
70     throws FileNotFoundException {
71     return makeOutputStream(new File(dir, filename), append);
72 }
73
74 public static OutputStream
75 makeOutputStream(File file, boolean append)
76     throws FileNotFoundException {
77     return new BufferedOutputStream(
78         new FileOutputStream(
79             file, append));
80 }
81
82 public static InputStream
83 makeInputStream(File file)
84     throws FileNotFoundException {
85     return new BufferedInputStream(new FileInputStream(file));
86 }
87
88
89 public static InputStream
90 makeInputStream(String filename)

```

```

91         throws FileNotFoundException {
92     return makeInputStream(new File(filename));
93 }
94
95 public static String readFileToString(File file) throws IOException {
96     FileInputStream stream = new FileInputStream(file);
97     try {
98         FileChannel fc = stream.getChannel();
99         MappedByteBuffer bb = fc.map(FileChannel.MapMode.READ_ONLY, 0, fc
100             .size());
101         return Charset.defaultCharset().decode(bb).toString();
102     } finally {
103         stream.close();
104     }
105 }
106
107 private static final int BUFFERSIZE = 128 * 1024;
108 private static byte[] buffer = new byte[BUFFERSIZE];
109 }

```

Python-komponentit (koodingenerointi)

Hakemistossa script/.

benchmark_generator.py

```
1 from templates import java_benchmark
2 from templates import java_counterparts
3 from templates import c_module
4 from templates import c_jni_function
5 from templates import c_nativemethod
6
7 from templating import put
8
9 import itertools
10 import logging
11
12 import jni_types
13 from jni_types import types, primitive_types, return_types
14
15 packagename = [
16     'fi',
17     'helsinki',
18     'cs',
19     'tituomin',
20     'nativebenchmark',
21     'benchmark']
22 library_name = 'nativebenchmark'
23 java_counterpart_classname = 'JavaCounterparts'
24 native_method_name = 'nativemethod'
25 class_counter = 0
26
27 # todo: initialize complex return values for c side
28
29
30 def next_sequence_no():
31     global class_counter
32     class_counter += 1
33     return str(class_counter).zfill(5)
34
35
36 def benchmark_classname(prefix, number):
37     return prefix + 'Benchmark' + number
38
39
40 def parameter_initialisation(language, typespec, name):
41     if typespec.get('is-array', False):
42         if language == 'java':
43             expression = 'benchmarkParameter.retrieve{_type}Array()'.format(
44                 _type=typespec['java-element-type'].capitalize())
```

```

45         else:
46             expression = '{_type}ArrayValue'.format(
47                 _type=typespec['c-element-type'])
48
49     elif typespec.get('is-object', False):
50         if language == 'java':
51             expression = 'benchmarkParameter.retrieve{_type}()'.format(
52                 _type=typespec['java'])
53         else:
54             expression = '{_type}Value'.format(
55                 _type=typespec['c'])
56
57     elif language == 'java':
58         if typespec.get('java-literal'):
59             expression = typespec['java-literal']
60         else:
61             expression = ''
62
63     elif language == 'c':
64         if typespec.get('c-literal'):
65             expression = typespec['c-literal']
66         else:
67             expression = ''
68
69     if name:
70         return name + " = " + expression
71     else:
72         return expression
73
74
75 def modifier_combinations():
76     privacy = ['public', 'private', 'protected']
77     static = ['static', '']
78     return list(itertools.product(privacy, static))
79
80
81 def method_combinations():
82     combinations = []
83     combinations.append({
84         'description': 'no arguments or return types',
85         'representative': True,
86         'return_types': [return_types['v']],
87         'target_modifiers': modifier_combinations(),
88         'types': []})
89
90     for symbol, _type in types.iteritems():
91         combinations.append({
92             'description': 'varying count {0}'.format(symbol),
93             'representative': _type.get('representative', False),
94             'return_types': [return_types['v']],
95             'target_modifiers': [('public', '')],
96             'types': jni_types.type_combinations(
97                 size=20,

```

```

98         typeset=[types[symbol]]))
99
100     # Start from 1 to avoid duplicates.
101     combinations.append({
102         'description': 'vary number of types',
103         'representative': True,
104         'skip': 1,
105         'return_types': [return_types['v']],
106         'target_modifiers': [('public', '')],
107         'types': jni_types.type_combinations(
108             typeset=types.values())
109     })
110
111     combinations.append({
112         'description': 'modifier combinations',
113         'representative': True,
114         'return_types': [return_types['l']],
115         'target_modifiers': modifier_combinations(),
116         'include_nonvirtual': True,
117         'types': [types['i']]
118     })
119
120     filtered_return_types = filter(
121         lambda x: x['symbol'] != 'l',
122         jni_types.type_combinations(typeset=types.values()))
123
124     combinations.append({
125         'description': 'return types',
126         'return_types': filtered_return_types,
127         'target_modifiers': [('public', '')],
128         'types': [types['i']]
129     })
130
131     return combinations
132
133
134 def generate_benchmarks():
135     global class_counter
136     java = []
137     java_callees = {}
138     c_implementations = []
139     c_runners = []
140     c_methodid_inits = []
141
142     for spec in method_combinations():
143         virtualities = [True]
144         if spec.get('include_nonvirtual'):
145             virtualities.append(False)
146         for virtualcall in virtualities:
147             for target_modifier in spec['target_modifiers']:
148                 for return_type in spec['return_types']:
149
150                     if 'representative' not in spec:

```

```

151         representative = return_type.get(
152             'representative', False)
153     else:
154         representative = spec['representative']
155
156     representative = "true" if representative else "false"
157
158     type_combination = spec['types']
159
160     native_method_modifiers = " ".join(target_modifier)
161     return_expression = parameter_initialisation(
162         'c', return_type, None)
163
164     # Declare/initialize the parameters that are passed
165     # to the called function/method.
166
167     parameter_names = []
168
169     parameter_declarations = []
170     parameter_initialisations = []
171     c_parameter_declarations = []
172     c_parameter_initialisations = []
173
174     if target_modifier[1] == 'static':
175         c_parameter_declarations.append('jclass cls')
176     else:
177         c_parameter_declarations.append('jobject instance')
178
179     for i, type_data in enumerate(type_combination):
180         parameter_names.append(
181             type_data['symbol'] + str(i + 1))
182
183         parameter_declarations.append(
184             type_data['java'] + ' ' + parameter_names[-1])
185         c_parameter_declarations.append(
186             type_data['c'] + ' ' + parameter_names[-1])
187         parameter_initialisations.append(
188             parameter_initialisation(
189                 'java', type_data, parameter_names[-1])){
190         c_parameter_initialisations.append(
191             parameter_initialisation(
192                 'c', type_data, parameter_names[-1]))
193
194     skip = spec.get('skip', 0)
195     upper_bound = len(type_combination){
196     if upper_bound == 0:
197         upper_bound = 1
198     for i in range(skip, upper_bound):
199
200         sequence_no = next_sequence_no()
201
202         for from_lang, to_lang in [
203             ('J', 'C'), ('J', 'J'),

```

```

204         ('C', 'C'), ('C', 'J')]:
205
206     pairing = (from_lang, to_lang)
207
208     if virtualcall == False:
209         # Nonvirtual calls only apply to C -> J
210         # instance methods J2C is generated as a proxy
211         # to get benchmark metadata
212         if (target_modifier[1] == 'static' or
213             pairing in [('J', 'J'), ('C', 'C')]):
214             continue
215
216     # 1. Set up call targets.
217
218     if to_lang == 'C':
219         counterpart_method_name = 'nativemethod'
220     if to_lang == 'J':
221         counterpart_method_unqualified = (
222             'benchmark' + sequence_no)
223
224     if pairing == ('J', 'J'):
225         if target_modifier[0] == 'private':
226             # A private method cannot
227             # be called from Java.
228             # Don't generate a benchmark.
229             continue
230
231         if target_modifier[1] == 'static':
232             clsname = java_counterpart_classname
233         else:
234             clsname = 'counterpartInstance'
235         counterpart_method_name = (
236             clsname + '.' +
237             counterpart_method_unqualified)
238         native_method = ''
239
240     if pairing == ('C', 'J'):
241         counterpart_method_name = 'benchmark' + \
242             sequence_no
243
244     # 2. Set up calling sources.
245
246     if from_lang == 'C':
247         run_method = java_benchmark.native_run_method_t
248         native_method = ''
249
250     if from_lang == 'J':
251         run_method = put(
252             java_benchmark.java_run_method_t,
253             parameter_declarations="; ".join(
254                 parameter_declarations[0:i + 1]),
255             parameter_initialisations="; ".join(
256                 parameter_initialisations[0:i + 1]),

```



```

257         counterpart_method_name=(
258             counterpart_method_name),
259         counterpart_method_arguments=(
260             ", ".join(parameter_names[0:i + 1])))
261
262     if pairing == ('J', 'C'):
263         native_method = put(
264             java_benchmark.native_method_t,
265             modifiers=native_method_modifiers,
266
267             return_type=return_type[
268                 'java'],
269             name=native_method_name,
270             parameters=", ".join(
271                 parameter_declarations[0:i + 1]))
272
273     # 3. Common benchmark class wrapper for all
274     # benchmarks.
275
276     classname = benchmark_classname(
277         '2'.join(pairing), sequence_no)
278
279     nm = native_method if to_lang == 'C' else ''
280     java.append({
281         'filename': classname + ".java",
282         'class':
283             ' '.join(
284                 packagename) + "." + classname,
285         'path': '/' + '.join(packagename),
286         'code': put(
287             java_benchmark.t,
288             representative=representative,
289             imports='',
290             has_dynamic_parameters='false',
291             # todo
292             is_allocating='false',
293             is_nonvirtual=(
294                 'false' if virtualcall else 'true'),
295             _id=benchmark_classname(
296                 "", sequence_no),
297             description=spec[
298                 'description'],
299             seq_no=class_counter,
300             from_language=from_lang,
301             to_language=to_lang,
302             class_relations='',
303             packagename=' '.join(
304                 packagename),
305             classname=classname,
306             library_name=library_name,
307             run_method=run_method,
308             native_method=nm})
309

```

```

310         if (return_type.get('is-object') or
311             return_type.get('is-array')):
312             ret_expression = "{_type}Value".format(
313                 _type=return_type['c'])
314         else:
315             ret_expression = parameter_initialisation(
316                 'java', return_type, None)
317
318         # 4. Call target implementations.
319
320         if to_lang == 'J':
321             cmu = counterpart_method_unqualified
322             java_callees[cmu] = put(
323                 java_counterparts.counterpart_t,
324                 return_type=return_type[
325                     'java'],
326                 privacy=target_modifier[0],
327                 static=target_modifier[1],
328                 methodname=cmu,
329                 parameters=", ".join(
330                     parameter_declarations[0:i + 1]),
331                 return_expression=ret_expression
332             )
333
334         if pairing == ('J', 'C'):
335             c_implementations.append(
336                 put(c_nativemethod.t,
337                     return_type=return_type['c'],
338                     packagename=(
339                         '_').join(
340                             packagename),
341                     classname=classname,
342                     function=native_method_name,
343                     parameters=", ".join(
344                         c_parameter_declarations[0:i + 2]),
345                     return_expression=return_expression))
346
347         if pairing == ('C', 'C'):
348             jni_param_names = ['env', 'instance']
349             jni_param_names.extend(parameter_names)
350
351             c_runners.append(
352                 put(c_nativemethod.t_caller_native,
353                     packagename='_'.join(packagename),
354                     classname=classname,
355                     parameter_declarations="; ".join(
356                         c_parameter_declarations[1:i + 2]),
357                     parameter_initialisations="; ".join(
358                         c_parameter_initialisations[
359                             0:i + 1]),
360                     counterpart_method_name=(
361                         "Java_{_}_{_}_{_}".format(
362                             ('_').join(packagename),

```

```

363         ('J2CBenchmark' +
364         str(sequence_no)),
365         native_method_name),
366         counterpart_method_arguments=", ".join(
367         jni_param_names[0:i + 3])))
368
369     if pairing == ('C', 'J'):
370         if return_type.get('is-object',
371                             return_type.get(
372                                 'is-array', False)):
373             java_method_type = 'Object'
374         else:
375             java_method_type = return_type[
376                 'java'].capitalize()
377
378     arguments = ', '.join(parameter_names[0:i + 1])
379     if arguments != '':
380         arguments = ', ' + arguments
381
382     c_runners.append(
383         c_nativemethod.call_java_from_c(
384             static=(
385                 target_modifier[1] == 'static'),
386             nonvirtual=not virtualcall,
387             seq_no=class_counter,
388             packagename='_ '.join(packagename),
389             classname=classname,
390             parameter_declarations="; ".join(
391                 c_parameter_declarations[1:i + 2]),
392             parameter_initialisations="; ".join(
393                 c_parameter_initialisations[
394                     0:i + 1]),
395             java_method_type=java_method_type,
396             call_variant='', # todo test variants?
397             arguments=arguments))
398
399     c_methodid_inits.append(
400         put(c_module.mid_init_t,
401            seq_no=class_counter,
402            static=target_modifier[
403                1].capitalize(),
404            method_name=counterpart_method_name,
405            method_descriptor=(
406                jni_types.method_descriptor(
407                    return_type,
408                    type_combination[0:i + 1])))
409
410     ref_types = jni_types.object_types.values(
411     ) + jni_types.array_types.values()
412     jcp_decl = ''
413     jcp_init = ''
414     for _type in ref_types:
415         jcp_decl += "private static {_type1} {_type2}Value;\n".format(

```

```

416         _type1=_type['java'], _type2=_type['c'])
417     jcp_init += parameter_initialisation(
418         'java', _type, _type['c'] + 'Value') + ";\n"
419
420     java_counterparts_class = {
421         'filename': java_counterpart_classname + ".java",
422         'class': ''.join(
423             packagename) +
424         "." +
425         java_counterpart_classname,
426         'path': '/'.join(packagename),
427         'code': put(java_counterparts.t,
428                     packagename=''.join(packagename),
429                     imports='',
430                     return_value_declarations=jcp_decl,
431                     return_value_inits=jcp_init,
432                     counterpart_methods=''.join(java_callees.values()))}
433
434     c_file = put(
435         c_module.t,
436         jni_function_templates=''.join(c_implementations),
437         initialisers='')
438
439     c_runners_file = put(
440         c_module.t,
441         jni_function_templates=''.join(c_runners),
442         initialisers=put(
443             c_module.initialisers_t,
444             mid_inits=''.join(c_methodid_inits),
445             amount_of_methods=class_counter))
446
447     return {'java': java,
448            'java_counterparts': java_counterparts_class,
449            'c': c_file,
450            'c_runners': c_runners_file}

```

jni_types.py

```

1  import itertools
2
3  primitive_types = None
4  object_types = None
5  other_types = None
6  types = None
7  return_types = None
8
9  representative_types = None
10
11 array_types = None

```

```

12
13 primitive_type_definitions = [
14     {
15         'symbol': 'b',
16         'java': 'boolean',
17         'c': 'jboolean',
18         'c-literal': '1',
19         'java-literal': 'true',
20         'jvm-desc': 'Z',
21         'byte_count': '1'
22     },
23
24     {
25         'symbol': 'y',
26         'java': 'byte',
27         'c': 'jbyte',
28         'c-literal': "'a'",
29         'java-literal': '(byte)100',
30         'jvm-desc': 'B',
31         'byte_count': '1',
32         'representative': True,
33         # todo: same value?
34     },
35
36     {
37         'symbol': 'c',
38         'java': 'char',
39         'c': 'jchar',
40         'c-literal': '12',
41         'java-literal': "'\u0012'",
42         'jvm-desc': 'C',
43         'byte_count': '2'
44     },
45
46     {
47         'symbol': 's',
48         'java': 'short',
49         'c': 'jshort',
50         'c-literal': '101',
51         'java-literal': '(short)101',
52         'jvm-desc': 'S',
53         'byte_count': '2'
54     },
55
56     {
57         'symbol': 'i',
58         'java': 'int',
59         'c': 'jint',
60         'c-literal': '102',
61         'java-literal': '102',
62         'jvm-desc': 'I',
63         'representative': True,
64         'byte_count': '4'

```

```

65     },
66
67     {
68         'symbol': 'l',
69         'java': 'long',
70         'c': 'jlong',
71         'c-literal': '103',
72         'java-literal': '103',
73         'jvm-desc': 'J',
74         'representative': True,
75         'byte_count': '8'
76     },
77
78     {
79         'symbol': 'f',
80         'java': 'float',
81         'c': 'jfloat',
82         'c-literal': '104.1',
83         'java-literal': '104.1f',
84         'jvm-desc': 'F',
85         'representative': True,
86         'byte_count': '4'
87     },
88
89     {
90         'symbol': 'd',
91         'java': 'double',
92         'c': 'jdouble',
93         'c-literal': '105.1',
94         'java-literal': '105.1',
95         'jvm-desc': 'D',
96         'representative': True,
97         'byte_count': '8'
98     },
99 ]
100
101 object_type_definitions = [
102
103     {
104         'symbol': 'O',
105         'java': 'Object',
106         'package': 'java.lang',
107         'c': 'jobject',
108         'c-literal': None,
109         'java-literal': None,
110         'is-object': True,
111         'representative': True,
112         'jvm-desc': 'Ljava/lang/Object;'
113     },
114
115     {
116         'symbol': 'C',
117         'java': 'Class',

```

```

118         'package': 'java.lang',
119         'c': 'jclass',
120         'c-literal': None,
121         'java-literal': None,
122         'is-object': True,
123         'jvm-desc': 'Ljava/lang/Class;'
124     },
125
126     {
127         'symbol': 'S',
128         'java': 'String',
129         'package': 'java.lang',
130         'c': 'jstring',
131         'c-literal': None,
132         'java-literal': '"a string"',
133         'is-object': True,
134         'jvm-desc': 'Ljava/lang/String;'
135     },
136
137     {
138         'symbol': 'T',
139         'java': 'Throwable',
140         'package': 'java.lang',
141         'c': 'jthrowable',
142         'c-literal': None,
143         'java-literal': None,
144         'is-object': True,
145         'jvm-desc': 'Ljava/lang/Throwable;'
146     }
147 ]
148 ]
149
150 other_type_definitions = [
151
152     {
153         'symbol': 'v',
154         'java': 'void',
155         'c': 'void',
156         'c-literal': None,
157         'java-literal': None,
158         'representative': True,
159         'jvm-desc': 'V'
160     }
161 ]
162
163
164 def java_native_methodname(is_static, returntype, parametertypes):
165     ret = "_"
166     if is_static:
167         ret += "st_"
168     ret += types.get(returntype)['java'] + "_"
169     for t in parametertypes:
170         ret += types.get(t)

```

```

171
172
173 def java_native_methodsignature(is_static, returntype, parametertypes):
174     ret = "private"
175     if is_static:
176         ret += "static "
177
178     # todo here
179
180
181 def type_combinations(size=0, typeset=None):
182     if size == 0:
183         size = len(typeset)
184
185     return list(itertools.islice(itertools.cycle(typeset), 0, size))
186
187
188 def method_descriptor(return_type, parameter_types):
189     return "({parameters}){returndesc}".format(
190         parameters=''.join([td['jvm-desc'] for td in parameter_types]),
191         returndesc=return_type['jvm-desc'])
192
193
194 def init_types():
195     global primitive_types, object_types, other_types, types
196     global return_types, array_types, representative_types
197
198     primitive_types = dict([(typedef['symbol'], typedef)
199                             for typedef in primitive_type_definitions])
200     object_types = dict([(typedef['symbol'], typedef)
201                          for typedef in object_type_definitions])
202     other_types = dict([(typedef['symbol'], typedef)
203                        for typedef in other_type_definitions])
204
205     array_element_types = {}
206     array_element_types.update(primitive_types)
207     array_element_types['0'] = object_types['0']
208
209     # todo here
210     array_types = dict(
211         [
212             ('A' + key,
213              {'symbol': 'A' + key,
214               'java': tipe['java'] + '[]',
215               'package': tipe.get('package', None),
216               'c': tipe['c'] + 'Array',
217               'c-literal': None,
218               'java-literal': None,
219               'is-array': True,
220               'representative': tipe.get('representative', False),
221               'java-element-type': tipe['java'],
222               'c-element-type': tipe['c'],
223               'jvm-desc': '[' + tipe['jvm-desc']

```



```

224         })
225         for key, tipe in array_element_types.iteritems()]]
226
227     types = dict()
228     types.update(primitive_types)
229     types.update(object_types)
230     types.update(array_types)
231
232     return_types = dict()
233     return_types.update(types)
234     return_types.update(other_types)
235
236 #     types.update(other_types)
237
238 init_types()

```

make_benchmarks.py

```

1 from benchmark_generator import generate_benchmarks, packagename
2 from make_custom_benchmarks import write_custom_benchmarks
3 from templates import java_registry_init
4 from templating import put
5
6 import sys
7 from sys import argv
8 import os.path
9 import os
10 import logging
11
12 # Log everything, and send it to stderr.
13 logging.basicConfig(level=logging.DEBUG)
14
15
16 def write_benchmark(benchmark, java_output_dir):
17     java_output_path = os.path.join(
18         java_output_dir,
19         benchmark["path"])
20
21     try:
22         os.makedirs(java_output_path)
23     except OSError:
24         pass
25
26     java_output = open(
27         os.path.join(
28             java_output_path,
29             benchmark["filename"]), 'w')
30
31     java_output.write(benchmark["code"])

```

```

32
33
34 def write_benchmarks(c_output, c_runners_output, java_output_dir):
35     benchmarks = generate_benchmarks()
36
37     c_output.write(benchmarks['c'])
38     c_runners_output.write(benchmarks['c_runners'])
39
40     write_benchmark(benchmarks['java_counterparts'], java_output_dir)
41     for benchmark in benchmarks['java']:
42         write_benchmark(benchmark, java_output_dir)
43
44     return [benchmark['class'] for benchmark in benchmarks['java']]
45
46
47 def write_benchmark_initialiser(classes):
48     benchmark_inits = []
49
50     for _class in classes:
51         benchmark_inits.append(java_registry_init.inits(_class))
52
53     path = os.path.join(
54         java_output_dir,
55         'fi/helsinki/cs/tituomin/nativebenchmark',
56         'BenchmarkInitialiser.java')
57
58     init_output = open(path, 'w')
59     init_output.write(
60         put(java_registry_init.t,
61            register_benchmarks="\n".join(benchmark_inits)))
62
63
64 if __name__ == "__main__":
65     try:
66         argv.pop(0)
67         c_output_name = argv.pop(0)
68         c_run_output_name = argv.pop(0)
69         c_custom_output_name = argv.pop(0)
70         java_output_dir = argv.pop(0)
71         c_definition_filename = argv.pop(0)
72         java_definition_filename = argv.pop(0)
73
74         definition_files = {
75             'C': open(c_definition_filename),
76             'J': open(java_definition_filename)}
77
78         c_run_output = open(c_run_output_name, 'w')
79         c_output = open(c_output_name, 'w')
80
81         classes = (write_benchmarks(c_output, c_run_output, java_output_dir) +
82                    write_custom_benchmarks(
83                        definition_files,
84                        c_custom_output_name,

```

```

85             java_output_dir))
86
87     write_benchmark_initialiser(classes)
88     print(", ".join(classes))
89 except Exception as e:
90     logging.exception("Exception was thrown.")
91     sys.exit(1)
92 else:
93     sys.exit(0)

```

make_custom_benchmarks.py

```

1 import re
2 import logging
3 from os import path
4 from sys import argv
5 import sys
6
7 from templating import put
8
9 from templates import arrays
10 from templates import loop_code
11 from templates import c_nativemethod
12 from templates import java_benchmark
13 from templates import java_registry_init
14
15 import jni_types
16
17 # Log everything, and send it to stderr.
18 logging.basicConfig(level=logging.DEBUG)
19
20 MAX_ALLOC_REPETITIONS = 500
21
22 i = re.IGNORECASE
23 begin_re = re.compile('\s*//\s*@begin\s*', flags=i)
24 end_re = re.compile('\s*//\s*@end\s*', flags=i)
25 inits_re = re.compile('\s*//\s*@inits-end\s*', flags=i)
26 benchmark_re = re.compile('\s*//\s*@(\\S+)\s*')
27
28
29 def inits_block_end(line):
30     return inits_re.match(line)
31
32
33 def begins_block(line):
34     return begin_re.match(line)
35
36
37 def ends_block(line):

```

```

38     return end_re.match(line)
39
40
41 def is_benchmark_header(line):
42     return benchmark_re.match(line)
43
44
45 def parse_benchmark_header(line):
46     tokens = line.split()[1:]
47     b_properties = parse_properties(tokens[1:])
48     b_properties['id'] = tokens[0][1:]
49     return b_properties
50
51
52 def parse_properties(seq):
53     kvs = []
54     for s in seq:
55         splitted = s.split('=')
56         kvs.append((splitted[0], splitted[1]))
57     try:
58         return dict(kvs)
59     except ValueError as e:
60         print seq
61         print seq[0].split('=')
62         exit(1)
63
64
65 def abort_if_last(line):
66     if line == '':
67         logging.error("Invalid benchmark input file.")
68         exit(1)
69
70
71 def read_until(f, predicate, collect=None):
72     line = ''
73     while not predicate(line):
74         if collect is not None:
75             collect.append(line)
76             line = f.readline()
77             abort_if_last(line)
78     abort_if_last(line)
79     return line
80
81
82 def read_benchmarks(definition_files):
83     benchmarks = {}
84     for lang, f in definition_files.iteritems():
85         benchmarks[lang] = {'module': None, 'benchmarks': []}
86
87         module_start = []
88         inits = []
89         read_until(f, begins_block, collect=module_start)
90         read_until(f, inits_block_end, collect=inits)

```

```

91     benchmarks[lang]['inits'] = ''.join(inits)
92     benchmarks[lang]['module'] = ''.join(module_start)
93     line = read_until(f, is_benchmark_header)
94
95     while line != '':
96         bm_props = parse_benchmark_header(line)
97
98         bm_code = []
99         line = read_until(f,
100             lambda x: ends_block(x) or is_benchmark_header(x),
101             collect=bm_code)
102
103         bm_props['code'] = ''.join(bm_code)
104         benchmarks[lang]['benchmarks'].append(bm_props)
105
106         if ends_block(line):
107             break
108
109     add_field_and_array_benchmarks(benchmarks)
110     add_overhead_benchmarks(benchmarks)
111     return benchmarks
112
113 OVERHEAD_STEP = 2
114 OVERHEAD_STEPS = 11 # incl. zero
115 OVERHEAD_CODE_STATEMENT = "__a = (((__a * __a * __a) / __b) + __b) / __a;\n"
116
117
118 def add_overhead_benchmark(benchmarks, i, prefix, alloc):
119     overhead_code = []
120     for j in range(0, i):
121         overhead_code.append(OVERHEAD_CODE_STATEMENT)
122
123     benchmark = {
124         'code': ''.join(overhead_code),
125         'id': prefix + 'Overhead' + str(i).zfill(5),
126         'description': i
127     }
128
129     if alloc:
130         benchmark['alloc'] = True
131
132     c_b = benchmark.copy()
133     double_benchmark = benchmark.copy()
134     # double the amount of work for java (uses optimizations unlike c)
135     double_benchmark['code'] = ''.join(overhead_code + overhead_code)
136     j_b = double_benchmark
137     c_b['direction'] = 'cj'
138     j_b['direction'] = 'jj'
139     benchmarks['C']['benchmarks'].append(c_b)
140     benchmarks['J']['benchmarks'].append(j_b)
141
142
143 def add_overhead_benchmarks(benchmarks):

```

```

144     for i in range(1, OVERHEAD_STEPS * OVERHEAD_STEP, OVERHEAD_STEP):
145         for prefix, alloc in [('Alloc', True), ('Normal', False)]:
146             add_overhead_benchmark(benchmarks, i, prefix, alloc)
147     add_overhead_benchmark(benchmarks, 200, 'Warmup', False)
148
149
150 def macro_call(template, _type):
151     return template.format(
152         _type=_type['c'],
153         java_type_name=_type['java'].capitalize())
154
155
156 def make_id(template, _type):
157     return template.format(
158         _type=_type['java'].capitalize())
159
160
161 def add_field_and_array_benchmarks(benchmarks):
162     c = benchmarks['C']['benchmarks']
163     java = benchmarks['J']['benchmarks']
164
165     for _type in (
166         jni_types.primitive_types.values() +
167         [jni_types.object_types['O']]):
168         representative = _type.get('representative', False)
169
170         c.append({
171             'id': make_id('GetStatic{_type}Field', _type),
172             'representative': representative,
173             'direction': 'cj',
174             'code': macro_call(
175                 'GET_STATIC_TYPE_FIELD({_type}, {java_type_name});',
176                 _type))
177
178         java.append({
179             'id': make_id('GetStatic{_type}Field', _type),
180             'representative': representative,
181             'direction': 'jj',
182             'class_init':
183                 'public {} persistentValue;'.format(_type['java']),
184             'method_init': '{} localPersistentValue = {};'.format(
185                 _type['java'], _type.get('java-literal') or 'objectValue'),
186             'code':
187                 ("localPersistentValue = "
188                  "mockObject.{_ctype}StaticField;"
189                  ).format(
190                     _javatype=_type['java'],
191                     _ctype=_type['c']
192                 ),
193             'finished': 'persistentValue = localPersistentValue;'
194         })
195     # todo: separate inits from global inits
196     # and make side-effect real

```

```

197
198 c.append({
199     'direction': 'cj',
200     'representative': representative,
201     'id': make_id('SetStatic{_type}Field', _type),
202     'code': macro_call(
203         'SET_STATIC_TYPE_FIELD({_type}, {java_type_name});',
204         _type)})
205
206 java.append({
207     'id': make_id('SetStatic{_type}Field', _type),
208     'representative': representative,
209     'direction': 'jj',
210     'code':
211         "mockObject.{_ctype}StaticField = {_literal} ;".format(
212             _ctype=_type['c'],
213             _literal=_type.get('java-literal') or 'objectValue'
214         ),
215     })
216
217 c.append({
218     'id': make_id('Get{_type}Field', _type),
219     'direction': 'cj',
220     'representative': representative,
221     'code': macro_call(
222         'GET_TYPE_FIELD({_type}, {java_type_name});',
223         _type)})
224
225 java.append({
226     'id': make_id('Get{_type}Field', _type),
227     'representative': representative,
228     'class_init':
229         'public {} persistentValue;'.format(_type['java']),
230     'method_init': '{} localPersistentValue = {};'.format(
231         _type['java'], _type.get('java-literal') or 'objectValue'),
232     'direction': 'jj',
233     'code':
234         "localPersistentValue = mockObject.{_ctype}Field;".format(
235             _javatype=_type['java'],
236             _ctype=_type['c']
237         ),
238     'finished': 'persistentValue = localPersistentValue;'
239     })
240
241 c.append({
242     'id': make_id('Set{_type}Field', _type),
243     'direction': 'cj',
244     'representative': representative,
245     'code': macro_call(
246         'SET_TYPE_FIELD({_type}, {java_type_name});',
247         _type)})
248
249

```

```

250
251     java.append({
252         'id': make_id('Set{_type}Field', _type),
253         'representative': representative,
254         'direction': 'jj',
255         'code':
256             "mockObject.{_ctype}Field = {_literal} ;".format(
257                 _ctype=_type['c'],
258                 _literal=_type.get('java-literal') or 'objectValue'
259             ),
260     })
261
262
263     for _type in jni_types.primitive_types.values():
264         representative = _type.get('representative', False)
265         c.append({
266             'id': make_id('New{_type}Array', _type),
267             'representative': representative,
268             'direction': 'cj',
269             'vary': 'size',
270             'alloc': 'true',
271             'code': macro_call(
272                 'NEW_PRIMITIVE_ARRAY({_type}, {java_type_name});',
273                 _type)
274         })
275
276     # java
277
278     c.append({
279         'id': make_id('Get{_type}ArrayElements', _type),
280         'representative': representative,
281         'direction': 'cj',
282         'vary': 'size',
283         'code': macro_call(
284             ('GET_PRIMITIVE_ARRAY_ELEMENTS({_type}, {java_type_name});'
285              'RELEASE_PRIMITIVE_ARRAY_ELEMENTS'
286              '({_type}, {java_type_name});'),
287             _type))
288
289     c.append({
290         'vary': 'size',
291         'direction': 'cj',
292         'representative': representative,
293         'id': make_id('Get{_type}ArrayRegion', _type),
294         'code': macro_call(
295             'GET_PRIMITIVE_ARRAY_REGION({_type}, {java_type_name});',
296             _type))
297
298     c.append({
299         'vary': 'size',
300         'representative': representative,
301         'direction': 'cj',
302         'id': make_id('Set{_type}ArrayRegion', _type),

```



```

303         'code': macro_call(
304             'SET_PRIMITIVE_ARRAY_REGION({_type}, {java_type_name});',
305             _type))
306
307     c.append({
308         'vary': 'size',
309         'representative': representative,
310         'direction': 'cj',
311         'id': make_id('Get{_type}ArrayLength', _type),
312         'code': macro_call(
313             'GET_PRIMITIVE_ARRAY_LENGTH({_type});',
314             _type))
315
316     c.append({
317         'vary': 'size',
318         'representative': representative,
319         'direction': 'cc',
320         'id': make_id('ReadComplete{_type}Array', _type),
321         'code': put(
322             arrays.t_read,
323             declare_idx='jint idx;',
324             variable_in='%s__IN' % _type['c'],
325             array_variable='%s_buf__IN' % _type['c'],
326             element_literal=_type['c-literal']
327         ))
328
329     java_declarations = ('{0} {0}In;\n{0} [] {0}Arr = '
330                          'benchmarkParameter.retrieve{1}Array();').format(
331         _type['java'], _type['java'].capitalize())
332
333     java.append({
334         'vary': 'size',
335         'direction': 'jj',
336         'representative': representative,
337         'class_init': 'public int persistentValue;',
338         'method_init': 'int localPersistentValue = 0;',
339         'id': make_id('ReadComplete{_type}Array', _type),
340         'code': put(
341             arrays.t_read,
342             declare_idx='int idx;',
343             declare_variables=java_declarations,
344             variable_in='%sIn' % _type['java'],
345             array_variable='%sArr' % _type['java'],
346             element_literal=_type['java-literal']
347         ),
348         'finished': 'persistentValue = localPersistentValue;'
349     })
350
351     c.append({
352         'vary': 'size',
353         'direction': 'cc',
354         'representative': representative,
355         'id': make_id('WriteComplete{_type}Array', _type),

```

```

356         'code': put(
357             arrays.t_write,
358             declare_idx='jint idx;',
359             # todo: writing affects other tests?
360             array_variable='%s_buf__IN' % _type['c'],
361             element_literal=_type['c-literal']
362         )))
363
364     java.append({
365         'vary': 'size',
366         'direction': 'jj',
367         'representative': representative,
368         'class_init': 'public int persistentValue;',
369         'method_init': 'int localPersistentValue = 0;',
370         'id': make_id('WriteComplete{_type}Array', _type),
371         'code': put(
372             arrays.t_write,
373             declare_variables=java_declarations,
374             declare_idx='int idx;',
375             # todo: writing affects other tests?
376             array_variable='%sArr' % _type['java'],
377             element_literal=_type['java-literal']),
378         'finished': 'persistentValue = localPersistentValue;'
379     })
380
381     # NIO variations of array/buffer reading/writing
382     if _type['java'] == 'boolean':
383         # Not available for booleans
384         continue
385
386     if _type['java'] == 'byte':
387         uppercase_typename = ''
388     else:
389         uppercase_typename = _type['java'].title()
390
391     # Read with hardcoded type method
392     java.append({
393         'vary': 'size',
394         'direction': 'jj',
395         'representative': True,
396         'class_init': 'public int persistentValue;',
397         'method_init': put(
398             arrays.t_init_nio,
399             type_declarations=java_declarations),
400         'id': make_id('ReadComplete{_type}NioByteBuffer', _type),
401         'code': put(
402             arrays.t_read_nio,
403             declare_idx='int idx;',
404             variable_in='%sIn' % _type['java'],
405             array_variable='directByteBufferValue',
406             type_name=uppercase_typename,
407             element_literal=_type['java-literal']),
408         'finished': 'persistentValue = localPersistentValue;'

```

```

409     })
410     # Write with hardcoded type method
411     java.append({
412         'vary': 'size',
413         'direction': 'jj',
414         'representative': True,
415         'class_init': 'public int persistentValue;',
416         'method_init': put(
417             arrays.t_init_nio,
418             type_declarations=java_declarations),
419         'id': make_id('WriteComplete{_type}NioByteBuffer', _type),
420         'code': put(
421             arrays.t_write_nio,
422             declare_variables='',
423             declare_idx='int idx;',
424             array_variable='directByteBufferValue',
425             type_name=uppercase_typename,
426             element_literal=_type['java-literal']),
427         'finished': 'persistentValue = localPersistentValue;'
428     })
429
430     declaration = java_declarations + "\n"
431     if _type['java'] == 'byte':
432         array_variable = 'directByteBufferValue'
433     else:
434         # Views are only relevant for non-byte types.
435         declaration += ('java.nio.{0}Buffer bufferView = '
436             'directByteBufferValue.as{0}Buffer();').format(
437             uppercase_typename)
438         array_variable = 'bufferView'
439
440     # Bulk read through a typecast view buffer
441     java.append({
442         'vary': 'size',
443         'direction': 'jj',
444         'representative': True,
445         'class_init': 'public int persistentValue;',
446         'method_init': put(
447             arrays.t_init_nio,
448             type_declarations=declaration),
449         'id': make_id('ReadBulk{_type}NioByteBufferView', _type),
450         'code': put(
451             arrays.t_bulk_read,
452             array_variable=array_variable,
453             array_in='%sArr' % _type['java']),
454         'finished': 'persistentValue = localPersistentValue;'
455     })
456
457     # Bulk write through a typecast view buffer
458     java.append({
459         'vary': 'size',
460         'direction': 'jj',
461         'representative': True,

```

```

462         'class_init': 'public int persistentValue;',
463         'method_init': put(
464             arrays.t_init_nio,
465             type_declarations=declaration),
466         'id': make_id('WriteBulk{_type}NioByteBufferView', _type),
467         'code': put(
468             arrays.t_bulk_write,
469             array_variable=array_variable,
470             array_in='%sArr' % _type['java']),
471         'finished': 'persistentValue = localPersistentValue;'
472     })
473
474     if _type['java'] == 'byte':
475         continue
476
477     # Read through a typecast view buffer
478     java.append({
479         'vary': 'size',
480         'direction': 'jj',
481         'representative': True,
482         'class_init': 'public int persistentValue;',
483         'method_init': put(
484             arrays.t_init_nio,
485             type_declarations=declaration),
486         'id': make_id('ReadComplete{_type}NioByteBufferView', _type),
487         'code': put(
488             arrays.t_read_nio_as_view,
489             declare_idx='int idx;',
490             variable_in='%sIn' % _type['java'],
491             array_variable='bufferView',
492             type_name=uppercase_typename,
493             element_literal=_type['java-literal']),
494         'finished': 'persistentValue = localPersistentValue;'
495     })
496     # Write through a typecast view buffer
497     java.append({
498         'vary': 'size',
499         'direction': 'jj',
500         'representative': True,
501         'class_init': 'public int persistentValue;',
502         'method_init': put(
503             arrays.t_init_nio,
504             type_declarations=declaration),
505         'id': make_id('WriteComplete{_type}NioByteBufferView', _type),
506         'code': put(
507             arrays.t_write_nio_as_view,
508             declare_idx='int idx;',
509             array_variable='bufferView',
510             element_literal=_type['java-literal']),
511         'finished': 'persistentValue = localPersistentValue;'
512     })

```

```

515 def write_custom_benchmarks(
516     definition_files,
517     c_custom_output_name,
518     java_output_dir):
519     packagename = (
520         'fi',
521         'helsinki',
522         'cs',
523         'tituomin',
524         'nativebenchmark',
525         'benchmark')
526
527     all_benchmarks = read_benchmarks(definition_files)
528
529     out_c = open(c_custom_output_name, 'w')
530     out_c.write(all_benchmarks['C']['module'])
531
532     java_classes = {} # classname, contents
533
534     for lang, data in all_benchmarks.iteritems():
535         for benchmark in data['benchmarks']:
536
537             direction = benchmark['direction']
538             from_lang, to_lang = direction[0].upper(), direction[1].upper()
539             if from_lang != lang:
540                 logging.error("Invalid language spec.")
541                 exit(1)
542
543             classname = '{0}2{1}'.format(from_lang, to_lang) + benchmark['id']
544             if 'vary' in benchmark:
545                 dyn_par = 'true'
546             else:
547                 dyn_par = 'false'
548             if 'alloc' in benchmark:
549                 # large heap 128/2 = 64 Mb, 128 el 8 byte array...
550                 is_allocating = 'true'
551             else:
552                 is_allocating = 'false'
553
554             representative = benchmark.get('representative', True)
555
556             if representative:
557                 representative = "true"
558             else:
559                 representative = "false"
560
561             if from_lang == 'C':
562                 out_c.write(put(
563                     c_nativemethod.t_run_method,
564                     return_type='void',
565                     parameters='jobject instance',
566                     function='runInternal',
567                     packagename='_'.join(packagename),

```

```

568         classname=classname,
569         body=put(
570             loop_code.t_c_jni_call,
571             debug=classname,
572             benchmark_body=benchmark[ 'code' ])))
573
574     java_classes[classname] = {
575         'filename': classname + '.java',
576         'code': (put(
577             java_benchmark.t,
578             representative=representative,
579             _id=benchmark[ 'id' ],
580             packagename='.'.join(packagename),
581             classname=classname,
582             description=benchmark.get( 'description', '' ),
583             is_allocating=is_allocating, # todo: measure
584             from_language=from_lang,
585             to_language=to_lang,
586             seq_no='-1',
587             has_dynamic_parameters=dyn_par,
588             is_nonvirtual='false',
589             run_method='public native void runInternal();',
590             ))}
591
592     elif from_lang == 'J' and to_lang == 'J':
593         java_classes[classname] = {
594             'filename': classname + '.java',
595             'code': (
596                 put(
597                     java_benchmark.t,
598                     representative=representative,
599                     _id=benchmark[ 'id' ],
600                     packagename='.'.join(packagename),
601                     imports="\n".join(
602                         [ 'import android.content.pm.PermissionInfo;',
603                           'import java.nio.ByteBuffer;',
604                           'import java.lang.ref.WeakReference;'
605                         ] ),
606                     classname=classname,
607                     class_fields=benchmark.get( 'class_init', '' ),
608                     description=benchmark.get( 'description', '' ),
609                     is_allocating=is_allocating,
610                     from_language=from_lang,
611                     to_language=to_lang,
612                     is_nonvirtual='false',
613                     seq_no='-1',
614                     has_dynamic_parameters=dyn_par,
615                     run_method=put(
616                         java_benchmark.java_run_method_inline_t,
617                         init=data[ 'inits' ],
618                         type_init=benchmark.get( 'method_init', '' ),
619                         loop=put(
620                             loop_code.t_java,

```

```

621         finished=benchmark.get('finished', ''),
622         benchmark_body=benchmark['code'])))})
623
624     out_c.flush()
625     out_c.close()
626
627     for classname, contents in java_classes.iteritems():
628         f = open(
629             path.join(java_output_dir,
630                 '/' + package_name,
631                 contents['filename']),
632             'w')
633         f.write(contents['code'])
634         f.flush()
635         f.close()
636
637     return ([ '/' + package_name + (classname,)
638             for classname in java_classes.keys()])

```

templates/arrays.py

```

1  from templating import partial
2
3  t_loop = """
4  <% declare_idx %>
5  <% declare_variables %>
6  for (idx = 0; idx < current_size; idx++) {
7      <% body %>
8  }
9  """
10
11 t_read = partial(
12     t_loop,
13     body="""
14     <% variable_in %> = <% array_variable %>[idx];
15     """)
16
17 t_write = partial(
18     t_loop,
19     body="""
20     <% array_variable %>[idx] = <% element_literal %>; """
21 )
22 t_init_nio = """
23     <% type_declarations %>
24     int localPersistentValue = 0;
25     current_size /= 64;
26
27 """
28

```

```

29 t_read_nio = partial(
30     t_loop,
31     body="""
32     <% variable_in %> = <% array_variable %>.get<% type_name %>(idx);
33     """)
34
35 t_write_nio = partial(
36     t_loop,
37     body="""
38     <% array_variable %>.put<% type_name %>(idx, <% element_literal %>);
39     """)
40
41 t_read_nio_as_view = partial(
42     t_loop,
43     body="""
44     <% variable_in %> = <% array_variable %>.get(idx);
45     """)
46
47 t_write_nio_as_view = partial(
48     t_loop,
49     body="""
50     <% array_variable %>.put(idx, <% element_literal %>);
51     """)
52
53 t_bulk_read = """
54 <% declare_variables %>
55 <% array_variable %>.clear();
56 <% array_variable %>.get(<% array_in %>, 0, current_size);
57 """
58
59 t_bulk_write = """
60 <% declare_variables %>
61 <% array_variable %>.clear();
62 <% array_variable %>.put(<% array_in %>, 0, current_size);
63 """

```

templates/c_jni_function.py

```

1 t = """
2
3 JNIEXPORT <% return_type %> JNICALL
4 Java_<% package %>_<% class_name %>_<% function %>
5 (JNIEnv *env, <% parameters %>) <%
6     <% set_returnvalues %>
7     %>
8
9 """

```



```

1 t = """
2 #include <jni.h>
3 #include <android/log.h>
4 #include <stdio.h>
5 #include "natives.h"
6 #include "native_benchmarks.h"
7 #include "returnvalues.h"
8
9 <% initialisers %>
10 <% jni_function_templates %>
11
12 """
13
14 initialisers_t = """
15 static jmethodID mids[<% amount_of_methods %>];
16
17 static void init_methodids(JNIEnv *env) {
18     jmethodID mid;
19     <% mid_inits %>
20 }
21
22
23 int check_interrupted(JNIEnv *env) {
24     jobject current_thread = NULL;
25     current_thread = (
26         (*env)->CallStaticObjectMethod(env, thread_class, current_thread_mid));
27     if (current_thread == NULL) {
28         __android_log_write(ANDROID_LOG_ERROR, "check_interrupted",
29             "Can't get current thread");
30         return 1;
31     }
32     jboolean interrupted = (*env)->CallBooleanMethod(
33         env, current_thread, is_interrupted_mid);
34     (*env)->DeleteLocalRef(env, current_thread);
35     if (interrupted == JNI_TRUE) {
36         return 1;
37     }
38     return 0;
39 }
40
41 void throw_interrupted_exception(JNIEnv *env) {
42     jclass newExcCls;
43     newExcCls = (*env)->FindClass(env,
44         "java/lang/InterruptedException");
45     if (newExcCls == NULL) {
46         /* Unable to find the exception class, give up. */
47         return;
48     }
49     (*env)->ThrowNew(env, newExcCls, "thrown from C code");
50 }

```

```

51
52 JNIEXPORT void JNICALL
53 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_initNative
54 (JNIEnv *env, jclass cls, jlong reps, jlong interval, jclass javaCounterparts,
55 jobject javaCounterpartsInstance, jclass thread_cls) {
56     repetitions = reps;
57     interrupted = 0;
58
59     CHECK_INTERRUPTED_INTERVAL = interval;
60
61     jclass java_counterparts_class_global_ref = NULL;
62     jclass thread_class_global_ref = NULL;
63     jobject java_counterparts_object_global_ref = NULL;
64
65     java_counterparts_class_global_ref = (*env)->NewGlobalRef(
66         env, javaCounterparts);
67     if (java_counterparts_class_global_ref == NULL) {
68         __android_log_write(ANDROID_LOG_ERROR, "initNative",
69             "Could not create global ref.");
70         return;
71     }
72     java_counterparts_class = java_counterparts_class_global_ref;
73
74     java_counterparts_object_global_ref = (*env)->NewGlobalRef(env,
75         javaCounterpartsInstance);
76     if (java_counterparts_object_global_ref == NULL) {
77         __android_log_write(ANDROID_LOG_ERROR, "initNative",
78             "Could not create global ref.");
79         return;
80     }
81     java_counterparts_object = java_counterparts_object_global_ref;
82
83     if (!(*env)->IsInstanceOf(env, java_counterparts_object,
84         java_counterparts_class)) {
85         __android_log_write(ANDROID_LOG_ERROR, "initNative",
86             "JavaCounterparts instance or class is not correct.");
87         return;
88     }
89
90     init_methodids(env);
91
92     thread_class_global_ref = (*env)->NewGlobalRef(env, thread_cls);
93     if (thread_class_global_ref == NULL) {
94         __android_log_write(ANDROID_LOG_ERROR, "initNative",
95             "Could not create global ref.");
96         return;
97     }
98     thread_class = thread_class_global_ref;
99
100     current_thread_mid = (*env)->GetStaticMethodID(env, thread_class,
101         "currentThread", "()Ljava/lang/Thread;");
102     if (current_thread_mid == NULL) {
103         __android_log_write(ANDROID_LOG_ERROR, "initNative",

```

```

104         "Could not find currentThread");
105     return;
106 }
107 is_interrupted_mid = (*env)->GetMethodID(env, thread_class,
108     "isInterrupted", "()Z");
109 if (is_interrupted_mid == NULL) {
110     __android_log_write(ANDROID_LOG_ERROR, "check_interrupted",
111         "Can't get isInterrupted method");
112     return;
113 }
114
115 }
116
117 JNIEXPORT void JNICALL
118 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_setRepetitions
119 (JNIEnv *env, jclass cls, jlong reps) {
120     repetitions = reps;
121 }
122
123 JNIEXPORT void JNICALL
124 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_interruptNative
125 (JNIEnv *env, jclass cls) {
126     interrupted = 1;
127 }
128
129 JNIEXPORT void JNICALL
130 Java_fi_helsinki_cs_tituomin_nativebenchmark_BenchmarkRegistry_resetInterruptFlag
131 (JNIEnv *env, jclass cls) {
132     interrupted = 0;
133 }
134
135 """
136
137 mid_init_t = """
138     mid = (*env)->Get<% static %>MethodID(
139         env, java_counterparts_class, "<% method_name %>",
140         "<% method_descriptor %>");
141     if (mid == NULL) {
142         __android_log_write(ANDROID_LOG_VERBOSE, "nativemethod",
143             "<% method_descriptor %> not found.");
144         return; /* method not found */
145     }
146     mids[<% seq_no %> - 1] = mid;
147
148 """

```

templates/c_nativemethod.py

```

1 from templating import partial, put
2 import loop_code
3
4 # todo: templates confusingly named
5
6 t_run_method = """
7 JNIEXPORT <% return_type %> JNICALL
8 Java_<% packagename %>_<% classname %>_<% function %>
9 (JNIEnv *env, <% parameters %>) {
10     <% parameter_declarations %>;
11     <% parameter_initialisations %>;
12     <% prebody %>
13     <% body %>
14 }
15
16 """
17
18 t = partial(
19     t_run_method,
20     body='return <% return_expression %>;',
21     remove=['parameter_declarations',
22             'parameter_initialisations',
23             'prebody'])
24
25 # C to C
26 t_caller_native = partial(
27     t_run_method,
28     return_type='void',
29     function='runInternal',
30     parameters='jobject instance',
31     prebody='',
32     body=partial(
33         loop_code.t_c_jni_call,
34         benchmark_body=(
35             '<% counterpart_method_name %>' +
36             '(<% counterpart_method_arguments %>);'))
37
38 # C to J
39 t_caller_java = partial(
40     t_run_method,
41     return_type='void',
42     function='runInternal',
43     parameters='jobject instance',
44     prebody='jmethodID mid = mids[<% seq_no %> - 1];')
45
46
47 def call_java_from_c(static=True, nonvirtual=False, **parameters):
48     benchmark_body = ''
49     if static:
50         benchmark_body = (
51             '(*env)->CallStatic<% java_method_type %>Method<% call_variant %>',
52             '(env, java_counterparts_class, mid<% arguments %>);')
53     elif nonvirtual:

```

```

54     benchmark_body = (
55         '(*env)->CallNonvirtual'
56         '<% java_method_type %>Method<% call_variant %>'
57         '(env, java_counterparts_object, java_counterparts_class,'
58         ' mid<% arguments %>);')
59     else:
60         benchmark_body = (
61             '(*env)->Call<% java_method_type %>Method<% call_variant %>' +
62             '(env, java_counterparts_object, mid<% arguments %>);')
63
64     parameters['body'] = put(
65         loop_code.t_c_jni_call,
66         benchmark_body=put(benchmark_body, **parameters))
67
68     return partial(t_caller_java, **parameters)

```

templates/__init__.py

1

templates/java_benchmark.py

```

1  from templating import partial
2  import loop_code
3  import logging
4
5  t = """
6  package <% packagename %>;
7
8  import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
9  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
10 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
11 import fi.helsinki.cs.tituomin.nativebenchmark.measuringtool.BasicOption;
12 import fi.helsinki.cs.tituomin.nativebenchmark.MockObject;
13 <% imports %>
14 import android.util.Log;
15
16 public class <% classname %> <% class_relations %> extends Benchmark {
17
18     public <% classname %> (BenchmarkParameter bp) {
19         init(bp);
20     }
21
22     public String from() {

```

```

23         return "<% from_language %>";
24     }
25
26     public String to() {
27         return "<% to_language %>";
28     }
29
30     public int sequenceNo() {
31         return <% seq_no %>;
32     }
33
34     public String id() {
35         return "<% _id %>";
36     }
37
38     public boolean representative() {
39         return <% representative %>;
40     }
41
42     public boolean dynamicParameters() {
43         return <% has_dynamic_parameters %>;
44     }
45
46     public String description() {
47         return "<% description %>";
48     }
49
50     public boolean isAllocating() {
51         return <% is_allocating %>;
52     }
53
54     public boolean isNonvirtual() {
55         return <% is_nonvirtual %>;
56     }
57
58     <% class_fields %>
59
60     <% native_method %>
61
62     <% run_method %>
63
64 }
65
66 """
67
68 native_method_t = ('<% modifiers %> native <% return_type %> '
69                  '<% name %> (<% parameters %>);')
70 dynamic_parameter_t = (
71     'new BasicOption(BasicOption.VARIABLE, "<% variable %>").strip()
72 native_run_method_t = 'public native void runInternal();'
73
74 loop = partial(loop_code.t_java,
75                benchmark_body=('<% counterpart_method_name %> '

```

```

76                                     '(<% counterpart_method_arguments %>);'))
77
78 java_run_method_t    = partial("""
79
80     public void runInternal() {
81         JavaCounterparts counterpartInstance = JavaCounterparts.INSTANCE;
82         <% parameter_declarations %>;
83         <% parameter_initialisations %>;
84
85         <% loop %>
86     }
87
88 """, loop=loop)
89
90 java_run_method_inline_t = """
91
92     public void runInternal() {
93         <% init %>
94         <% type_init %>
95         <% loop %>
96     }
97
98 """

```

templates/java_counterparts.py

```

1  from templating import put
2
3  t = """
4  package <% packagename %>;
5
6  <% imports %>
7  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
8  import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRunner;
9  import android.util.Log;
10
11
12  public enum JavaCounterparts {
13      INSTANCE;
14
15      <% return_value_declarations %>
16      public int persistentValue;
17      public static int staticpersistentValue = 0;
18
19      private JavaCounterparts() {
20          persistentValue = 0;
21      }
22
23      public static void initParams(BenchmarkParameter benchmarkParameter) {

```

```

24         <% return_value_inits %>
25     }
26
27     <% counterpart_methods %>
28
29 }
30
31 """
32
33 # todo         <% return_values %>
34
35
36 counterpart_t = """
37
38 <% privacy %> <% static %> <% return_type %> <% methodname %> (<% parameters %>) {
39     <% static %> persistentValue = (<% static %> persistentValue + 1) % 10;
40     return <% return_expression %>;
41 }
42
43 """
44
45 return_value_t = (
46     "private static <% actualtype %> = "
47     "benchmarkParameter.retrieve<% typename %> (<% typespecs %>);")

```

templates/java_registry_init.py

```

1 from templating import put
2
3 t = """
4 package fi.helsinki.cs.tituomin.nativebenchmark;
5
6 import fi.helsinki.cs.tituomin.nativebenchmark.Benchmark;
7 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkRegistry;
8 import fi.helsinki.cs.tituomin.nativebenchmark.BenchmarkParameter;
9 import fi.helsinki.cs.tituomin.nativebenchmark.benchmark.*;
10 import java.util.List;
11
12 public class BenchmarkInitialiser {
13
14     public static void init(BenchmarkParameter bp) {
15         List<Benchmark> benchmarks = BenchmarkRegistry.getBenchmarks();
16
17         <% register_benchmarks %>
18     }
19
20 }
21
22 """

```



```

23
24
25 def inits(classname):
26     return 'benchmarks.add(new {0} (bp));'.format(classname)

```

templates/loop_code.py

```

1 from templating import put, partial
2
3 t = """
4     <% declare_counters %>
5     <% additional_declaration %>
6
7     <% init_counters %>
8     division = repetitions / interval + 1;
9     remainder = repetitions % interval + 1;
10
11     <% debug %>
12     <% additional_init %>
13     while (--division != 0) {
14         <% init_counters %>
15         interval = interval + 1;
16         while (--interval != 0) {
17             <% pre_body %>
18             <% extra_debug %>
19             <% benchmark_body %>
20             <% post_body %>
21         }
22         if (<% test_interrupted %>) {
23             <% debug_interrupted %>
24             return;
25         }
26     }
27
28     <% additional_init %>
29
30     while (--remainder != 0) {
31         <% pre_body %>
32         <% benchmark_body %>
33         <% post_body %>
34     }
35
36     <% removal_prevention %>
37     <% finished %>
38
39 """
40
41 jni_push_frame = """
42 if (refs == 0) {

```

```

43     refs = LOCAL_FRAME_SIZE;
44     if ((*env)->PushLocalFrame(env, LOCAL_FRAME_SIZE) < 0) {
45         return;
46     }
47 }
48 """
49
50 jni_pop_frame = """
51 if (--refs == 0) {
52     (*env)->PopLocalFrame(env, NULL);
53 }
54
55 """
56
57
58 t_c_base = partial(
59     t,
60     declare_counters='jlong interval, division, remainder;',
61     init_counters='interval = CHECK_INTERRUPTED_INTERVAL;',
62     test_interrupted='interrupted')
63
64 t_c_jni_call = partial(
65     t_c_base,
66     additional_declaration='jlong refs;',
67     additional_init='refs = 0;',
68     remove=[
69         'extra_debug',
70         'debug',
71         'debug_interrupted',
72         'removal_prevention'],
73     pre_body=jni_push_frame,
74     post_body=jni_pop_frame)
75
76 t_c = partial(
77     t_c_base,
78     remove=['extra_debug', 'debug', 'debug_interrupted',
79             'additional_declaration', 'additional_init',
80             'pre_body', 'post_body', 'removal_prevention'])
81
82 t_java = partial(
83     t,
84     test_interrupted='Thread.currentThread().isInterrupted()',
85     extra_debug='', # , 'Log.v("Benchmark", division + " " + interval);',
86     declare_counters='long interval, division, remainder;',
87     init_counters='interval = BenchmarkRegistry.CHECK_INTERRUPTED_INTERVAL;',
88     removal_prevention='repetitionsLeft = division * interval + remainder;',
89     remove=['additional_declaration',
90             'additional_init',
91             'pre_body',
92             'post_body'])

```

```
1 import string
2 import logging
3 formatter = string.Formatter()
4
5
6 class PartialDict(dict):
7
8     def __missing__(self, key):
9         return "<% " + key + " %>"
10
11
12 class PurgeDict(dict):
13
14     def __missing__(self, key):
15         return ""
16
17
18 def escape(string):
19     string = string.replace('{', '__BEG__')
20     string = string.replace('}', '__END__')
21     string = string.replace('<% ', '{')
22     string = string.replace(' %>', '}')
23     return string
24
25
26 def unescape(string):
27     string = string.replace('{', '<% ')
28     string = string.replace('}', ' %>')
29     string = string.replace('__BEG__', '{')
30     string = string.replace('__END__', '}')
31     return string
32
33
34 def put(template, remove=None, purge=True, **kwargs):
35     try:
36         template = escape(template)
37         for k, v in kwargs.iteritems():
38             if isinstance(v, str):
39                 kwargs[k] = escape(v)
40             if v is None:
41                 kwargs[k] = ''
42
43         if remove:
44             for k in remove:
45                 kwargs[k] = ''
46
47         if purge:
48             fdict = PurgeDict(**kwargs)
49         else:
50             fdict = PartialDict(**kwargs)
```

```
51
52     result = formatter.vformat(template, (), fdict)
53     result = unescape(result)
54     return result
55 except ValueError as e:
56     logging.error('error with template: ' + template)
57     raise e
58 return None
59
60
61 def partial(template, remove=None, **kwargs):
62     return put(template, remove=remove, purge=False, **kwargs)
```

Python-komponentit (analyysi)

Hakemistossa .

analysis.py

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from numpy import polyfit, reshape, polyval
5
6 def linear_fit_columns(x, y):
7     p, residuals, rank, singular_values, rcond = polyfit(x, y, 1, full=True)
8     ynorm = normalized(x, y, p)
9     pnorm, residuals, rank, singular_values, rcond = polyfit(
10         x, ynorm, 1, full=True)
11     return p, residuals
12
13 def normalized(x, y, poly):
14     # normali
15     return (x - poly[1]) / poly[0]
16
17 def linear_fit(rows):
18     columns = reshape(rows, len(rows)*len(rows[0]), order='F').reshape(
19         (len(rows[0]), -1))
20     x = columns[0]
21     columns = columns[1:]
22     residuals = [linear_fit_columns(x, col)[1][0] for col in columns]
23     polys = [linear_fit_columns(x, col)[0] for col in columns]
24     return x, polys, residuals
25
26 def estimate_measuring_overhead(rows):
27     x, polys, residuals = linear_fit(rows)
28     return [p[1] for p in polys]
29
30 def optimize_bins(x):
31     """
32     Created on Thu Oct 25 11:32:47 2012
33
34     Histogram Binwidth Optimization Method
35
36     Shimazaki and Shinomoto, Neural Comput 19 1503-1527, 2007
37     2006 Author Hideaki Shimazaki, Matlab
38     Department of Physics, Kyoto University
39     shimazaki at ton.scphys.kyoto-u.ac.jp
40     Please feel free to use/modify this program.
41
42     Version in python adapted Érbet Almeida Costa
43
44     Data: the duration for eruptions of
```

```

45  the Old Faithful geyser in Yellowstone National Park (in minutes)
46  or normal distribution.
47  Version in python adapted Érbet Almeida Costa
48  Bugfix by Takuma Torii 2.24.2013
49
50  """
51
52  import numpy as np
53  from numpy import mean, size, zeros, where, transpose
54  from numpy.random import normal
55  from matplotlib.pyplot import hist
56  from scipy import linspace
57  import array
58
59  x_max = max(x)
60  x_min = min(x)
61  N_MIN = 4      #Minimum number of bins (integer)
62                #N_MIN must be more than 1 (N_MIN > 1).
63  N_MAX = 1000   #Maximum number of bins (integer)
64  N = range(N_MIN,N_MAX) # #of Bins
65  N = np.array(N)
66  D = (x_max-x_min)/N    #Bin size vector
67  C = zeros(shape=(size(D),1))
68
69  #Computation of the cost function
70  for i in xrange(size(N)):
71      edges = linspace(x_min,x_max,N[i]+1) # Bin edges
72      ki = hist(x,edges) # Count # of events in bins
73      ki = ki[0]
74      k = mean(ki) #Mean of event count
75      v = sum((ki-k)**2)/N[i] #Variance of event count
76      C[i] = (2*k-v)/((D[i])**2) #The cost Function
77  #Optimal Bin Size Selection
78
79  cmin = min(C)
80  idx  = where(C==cmin)
81  idx = int(idx[0])
82  optD = D[idx]
83
84  edges = linspace(x_min,x_max,N[idx]+1)
85
86  return optD, edges

```

datafiles.py

```

1  #!/usr/bin/python
2
3  import re
4  from collections import OrderedDict as odict

```

```

5 import sys
6
7 SEPARATOR = ','
8 RE_EMPTY = re.compile('^\\s*$')
9 RE_NUMERICAL = re.compile('^-[0-9]+$')
10
11
12 def explode(line):
13     return line.split(SEPARATOR)
14
15 def value(string, key=None):
16     if key in ['start', 'end']:
17         return string
18     if key == 'class':
19         return string.split('.')[ -1]
20     if string == '-' or RE_EMPTY.match(string):
21         return None
22     if RE_NUMERICAL.match(string):
23         return int(string)
24     else:
25         return string
26
27 def empty_label():
28     empty_label.cnt += 1
29     return 'empty_{0}'.format(empty_label.cnt)
30
31 empty_label.cnt = 0
32
33 def read_datafiles(files, silent=False):
34     if not silent:
35         print 'Reading from %s files' % len(files)
36     benchmarks = []
37     #-1: there is an empty field at the end...
38
39     keys_with_values = set()
40     all_keys = set()
41
42     lineno = 1
43     for i, f in enumerate(files):
44         line = f.readline()
45         labels = explode(line)
46         for i, l in enumerate(labels):
47             # account for the fact that there might be an empty label
48             # and corresponding column (usually the last)
49             if RE_EMPTY.match(l):
50                 labels[i] = empty_label()
51
52         all_keys.update(labels)
53
54         line = f.readline()
55         while line != '':
56             exploded_line = explode(line)
57             pad_amount = len(labels) - len(exploded_line)

```

```

58         exploded_line.extend(['-'] * pad_amount)
59         if len(labels) != len(exploded_line):
60             print ('missing values', f.name, 'line', lineno, 'labels',
61                   len(labels), 'values', len(exploded_line))
62             exit(1)
63
64         benchmark = dict()
65         benchmark['lineno'] = lineno
66
67         for key, string in zip(labels, exploded_line):
68             benchmark[key] = value(string, key=key)
69
70             if value(string, key=key) != None:
71                 keys_with_values.add(key)
72
73             #if benchmark['response_time'] != None:
74                 benchmarks.append(benchmark)
75
76         line = f.readline()
77         lineno += 1
78
79     keys_without_values = all_keys - keys_with_values
80
81     benchmark_keycount = None
82     for benchmark in benchmarks:
83         for key in keys_without_values:
84             if key in benchmark:
85                 del benchmark[key]
86             current_keycount = len(benchmark.keys())
87             benchmark_keycount = benchmark_keycount or current_keycount
88             if benchmark_keycount != current_keycount:
89                 print ("Benchmarks have different amount of data",
90                       benchmark_keycount, current_keycount, "at line",
91                       benchmark['lineno'])
92                 exit(1)
93
94     if not silent:
95         print 'Read %d lines' % (lineno - 1)
96     return benchmarks
97
98 def read_measurement_metadata(mfile, combine_compatibles):
99     compatibles = odict()
100     measurement = None
101     line = None
102
103     i = 0
104     while line != '':
105         skipped = False
106         while line == "\n":
107             line = mfile.readline()
108             skipped = True
109
110         if skipped:

```



```

111         if measurement:
112             if 'tools' in measurement:
113                 measurement['tool'] = measurement['tools']
114                 revision = measurement.get('code-revision')
115                 checksum = measurement.get('code-checksum')
116                 repetitions = measurement.get('repetitions')
117                 tool = measurement.get('tool')
118                 cpufreq = measurement.get('cpu-freq')
119                 benchmark_set = measurement.get('benchmark-set')
120                 substring_filter = measurement.get('substring-filter')
121                 if measurement.get('rounds') == None:
122                     measurement['rounds'] = 1
123
124             if revision and repetitions:
125                 if combine_compatibles:
126                     key = (revision, checksum, repetitions, tool, cpufreq,
127                           benchmark_set, substring_filter)
128                 else:
129                     key = i
130                     i += 1
131                 if key not in compatibles:
132                     compatibles[key] = []
133                 compatibles[key].append(measurement)
134         measurement = {}
135
136     if line != None:
137         splitted = line.split()
138         if len(splitted) > 1:
139             key = splitted[0].rstrip(':')
140             val = ' '.join(splitted[1:])
141             measurement[key] = val.strip()
142
143     line = mfile.readline()
144
145     return compatibles

```

gnuplot.py

```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import os
5 import uuid
6
7 INIT_PALETTE = """
8 # line styles for ColorBrewer Dark2
9 # for use with qualitative/categorical data
10 # provides 8 dark colors based on Set2
11 # compatible with gnuplot >=4.2

```

```

12 # author: Anna Schneider
13
14 # line styles
15 set style line 1 pt 7 lt 1 lc rgb '#1B9E77' # dark teal
16 set style line 2 pt 7 lt 1 lc rgb '#D95F02' # dark orange
17 set style line 3 pt 7 lt 1 lc rgb '#7570B3' # dark lilac
18 set style line 4 pt 7 lt 1 lc rgb '#000000' # black
19 set style line 5 pt 7 lt 1 lc rgb '#E7298A' # dark magenta
20 set style line 6 pt 7 lt 1 lc rgb '#66A61E' # dark lime green
21 set style line 7 pt 7 lt 1 lc rgb '#E6AB02' # dark banana
22 set style line 8 pt 7 lt 1 lc rgb '#A6761D' # dark tan
23 set style line 9 pt 7 lt 1 lc rgb '#666666' # dark gray
24 set style line 10 pt 7 lt 1 lc rgb '#1b70b3' # dark blue
25
26 set style line 11 pt 5 lt 2 lc rgb '#1B9E77' # dark teal
27 set style line 12 pt 5 lt 2 lc rgb '#D95F02' # dark orange
28 set style line 13 pt 5 lt 2 lc rgb '#7570B3' # dark lilac
29 set style line 14 pt 5 lt 2 lc rgb '#000000' # black
30 set style line 15 pt 5 lt 2 lc rgb '#E7298A' # dark magenta
31 set style line 16 pt 5 lt 2 lc rgb '#66A61E' # dark lime green
32 set style line 17 pt 5 lt 2 lc rgb '#E6AB02' # dark banana
33 set style line 18 pt 5 lt 2 lc rgb '#A6761D' # dark tan
34 set style line 19 pt 5 lt 2 lc rgb '#666666' # dark gray
35 set style line 20 pt 5 lt 2 lc rgb '#1b70b3' # dark blue
36
37
38 # palette
39 set palette maxcolors 8
40 set palette defined ( 0 '#1B9E77',\
41                      1 '#D95F02',\
42                      2 '#7570B3',\
43                      3 '#E7298A',\
44                      4 '#66A61E',\
45                      5 '#E6AB02',\
46                      6 '#A6761D',\
47                      7 '#666666' )
48 ""
49
50 INIT_PLOTS_PDF = ""
51 set terminal pdfcairo size 32cm,18cm {sizesuffix}
52 set size 1, 0.95
53 set output '{filename}'
54 ""
55
56 INIT_PLOTS_LATEX = ""
57 set terminal epslatex input color \
58 header "\\|\\caption{{{caption}}}|\\|\\label{{fig:{label}}}" {sizesuffix}
59 set pointsize 1.0
60 set format y "%4.2s%cs"
61 set output
62 ""
63
64 INIT_PLOTS_SVG = ""

```

```

65 set terminal svg {sizesuffix}
66 set pointsize 1.0
67 set format y "%4.2s%cs"
68 set output
69 ""
70
71 INIT_PLOTS_COMMON = ""
72 set grid
73 set xlabel "kutsuparametrien määrä"
74 ""
75
76 INIT_PLOT_LABEL_PDF = ""
77 set label 1 "{bid}" at graph 0.01, graph 1.06
78 ""
79
80 TEMPLATES = {}
81 INIT_KEY = {}
82
83 TEMPLATES['binned_init'] = ""
84 set title '{title}'
85 binwidth={binwidth}
86 set boxwidth binwidth
87 set style fill solid 1.0
88 set xrange [{min_x}:{max_x}]
89 set yrange [0:{max_y}]
90 ""
91 # border lt -1
92 #bin(x,width)=width*floor(x/width) + width/2.0
93
94 TEMPLATES['binned_frame'] = ""
95 #set label 2 "{datapoints}" at graph 0.8, graph 1.06
96 set bmargin 20
97 set tmargin 20
98 set rmargin 20
99 set lmargin 20
100 plot '-' using 1:2 notitle with boxes lt rgb "{color}"\n{values}\ne\n
101 #unset xlabel
102 #unset ylabel
103 #unset label 1
104 #unset title
105 unset xtics
106 unset ytics
107 ""
108
109 SET_TITLE_AND_PAGE_LABEL = ""
110 set title '{title}'
111 set label 2 "{page}" at screen 0.9, screen 0.95
112 ""
113
114 INIT_KEY['simple_groups'] = ""
115 set key {key_placement} box notitle width -3 height +1 vertical
116 ""
117

```

```

118 TEMPLATES['simple_groups'] = """
119 set ylabel "vasteaika {reps} toistolla"
120 set xlabel "{xlabel}"
121 plot for [I=2:{last_column}] '{filename}' index {index} \
122 using 1:I title columnhead with points ls I-1
123 """
124
125 TEMPLATES['fitted_lines'] = """
126 set ylabel "vasteaika {reps} toistolla"
127 set xlabel "{xlabel}"
128 plot for [I=2:{last_real_column}] '{filename}' index {index} using 1:I \
129 title columnhead with points ls I-1, \
130 for [I={first_fitted_column}:{last_column}] '{filename}' index {index} \
131 using 1:I notitle with lines ls I-{first_fitted_column}+1
132 """
133
134 TEMPLATES['named_columns'] = """
135 set yrange [0:*]
136 set xlabel "{xlabel}"
137 plot for [I=2:{last_column}] '{filename}' index {index} using I:xtic(1) \
138 title columnhead with linespoints
139 """
140
141 TEMPLATES['histogram'] = """
142 #set xlabel "{xlabel}"
143 unset xlabel
144 #set xlabel "{xlabel}" rotate
145 unset ylabel
146 set y2label "vasteaika {reps} toistolla"
147 set size 1, 1
148 unset x2tics
149 #unset xtics
150 unset ytics
151
152 set y2tics format "%.00s%cs" rotate
153
154 set xtics out rotate
155 set key at graph 0.1, 0.9 width 2 height 8 notitle horizontal nobox samplen 0.2
156 set label 1 'C$\rightarrow$Java' at graph 0.145, 0.78 left rotate by 90
157 set label 2 'Java$\rightarrow$Java' at graph 0.205, 0.78 left rotate by 90
158 # set label 2 'Nowhere' at graph 0.09, 0.85 left rotate by 90
159 # set label 3 'Everywhere' at graph 0.2, 0.85 left rotate by 90
160 #set boxwidth 0.9 relative
161 # set style fill solid border lc rgbcolor "black"
162 set style data histograms
163 set style histogram clustered
164 #set style fill solid 1.0 border lt -1
165
166 plot [] [0:*] for [I=2:{last_column}] '{filename}' index {index} \
167 using I:xtic(1) every ::1 title " " with histogram fillstyle solid 1.0 border lt -1
168 """
169
170 measurement_id = None

```

```

171 plot_directory = '/home/tituomin/gradu/paper/figures/plots'
172
173 def init(plotscript, filename, mid, output_type='pdf'):
174     global measurement_id, plot_directory
175     measurement_id = mid
176     if output_type == 'pdf':
177         plotscript.write(INIT_PLOTS_PDF.format(filename=filename))
178         plotscript.write(INIT_PLOT_LABEL_PDF.format(bid=measurement_id))
179     plotscript.write(INIT_PLOTS_COMMON)
180     plotscript.write(INIT_PALETTE)
181
182 GROUPTITLES={
183     'direction': 'kutsusuunta',
184     'from': 'kieli'
185 }
186
187 def output_plot(data_headers, data_rows, plotpath,
188                 plotscript, title, specs, style, page,
189                 identifier,
190                 xlabel, additional_data=None, output='pdf',
191                 key_placement="inside top left", reps='XXX-fixme-XXX'):
192     global plot_directory
193     template = TEMPLATES[style]
194
195     rowlen = len(data_rows[0]) - 1
196     size = 'normal'
197     if style == 'fitted_lines':
198         rowlen /= 2
199     if (page > 51 and rowlen > 7) or rowlen > 10:
200         size = 'tall'
201     if rowlen < 15:
202         size = 'normal'
203     if identifier in [
204         'basic-call-all-types-j-j-fit',
205         'basic-call-all-types-c-c-fit',
206         'variable-argument-size-j-c',
207         'special-calls-arrayelements-c-j-fit',
208         'special-calls-arrayregion-c-j-fit']:
209         size = 'tall'
210
211     if output in ['latex', 'svg']:
212         if output == 'latex':
213             init_tmpl = INIT_PLOTS_LATEX
214             file_suffix = 'tex'
215         elif output == 'svg':
216             init_tmpl = INIT_PLOTS_SVG
217             file_suffix = 'svg'
218         sizesuffix=''
219         if size == 'tall':
220             if output == 'svg':
221                 sizesuffix = 'size 1000,800'
222             else:
223                 sizesuffix="size 15cm,13cm"

```

```

224     else:
225         if output == 'svg':
226             sizesuffix="size 1000,600"
227         else:
228             sizesuffix="size 15cm,10cm"
229     plotscript.write(
230         init_tmpl.format(
231             caption=title,
232             label=identifier,
233             sizesuffix=sizesuffix))
234     if specs['variable'] == 'dynamic_size':
235         plotscript.write("set xrange [0:512]\n")
236         plotscript.write("set xtics 0, 64\n")
237         plotscript.write("set format x \"%6.sB\"\n")
238     else:
239         plotscript.write("unset xtics\n")
240         plotscript.write("set xtics autofreq\n")
241         plotscript.write("set xrange [*:*]\n")
242         plotscript.write("set format x \"%6.s\"\n")
243
244     if size == 'tall':
245         if identifier in ['special-calls-arrayelements-c-j-fit',
246                         'special-calls-arrayregion-c-j-fit']:
247             plotscript.write(
248                 "set tmargin at screen 0.8\nset key above box "
249                 "horizontal maxrows 8 maxcols 4 samplen 1 "
250                 "spacing .5 font \",4\"\n");
251         else:
252             plotscript.write(
253                 "set tmargin at screen 0.85\n"
254                 "set key above nobox horizontal\n");
255     else:
256         plotscript.write("set tmargin at screen 0.95\n")
257     plotscript.write("set output '{}'.format(
258         os.path.join(plot_directory,
259                     "plot-{}-{}.{}".format(
260                         measurement_id, identifier, file_suffix))))
261
262     if plotpath:
263         # external data
264         filename = os.path.join(plotpath, "plot-" + str(uuid.uuid4()) + ".data")
265         plotdata = open(filename, 'w')
266         specs['convert_to_seconds'] = False # (output == 'latex')
267         if output == 'latex':
268             specs['tinylabels'] = True
269         if output == 'svg':
270             specs['scriptlabels'] = True
271         plotdata.write(print_benchmarks(data_headers, data_rows, title,
272                                         **specs))
273
274     miny = 0
275     for row in data_rows:
276         for cell in row[1:]:

```

```

277         if cell < miny:
278             miny = cell
279     if miny == None:
280         miny = '*'
281
282     if output == 'pdf':
283         plotscript.write(SET_TITLE_AND_PAGE_LABEL.format(page=identifier,
284                                                         title=title))
285
286     if style == 'binned':
287         plotscript.write(template.format(
288             title = title, page = identifier, filename = filename, index = 0,
289             last_column = len(data_rows[0]),
290             xlabel = xlabel, miny=miny, **additional_data))
291
292     elif style == 'fitted_lines':
293         length = len(data_headers) - 1
294         last_real_column = 1 + length / 2
295         first_fitted_column = last_real_column + 1
296         plotscript.write(template.format(
297             title = title, reps = reps, page = identifier, filename = filename,
298             index = 0, last_column = len(data_rows[0]),
299             xlabel = xlabel, miny=miny, last_real_column=last_real_column,
300             first_fitted_column=first_fitted_column))
301
302     elif style == 'simple_groups':
303         grouptitle = GROUPTITLES.get(specs['group'], 'group')
304         if key_placement is None:
305             plotscript.write("\nunset key\n")
306         elif size != 'tall':
307             plotscript.write(INIT_KEY[style].format(
308                 key_placement=key_placement))
309
310         plotscript.write(template.format(
311             title = title, reps = reps, page = identifier, filename = filename,
312             index = 0, last_column = len(data_rows[0]),
313             xlabel = xlabel, miny=miny, grouptitle=grouptitle))
314
315     else:
316         grouptitle = GROUPTITLES.get(specs['group'], 'group')
317         plotscript.write(template.format(
318             title = title, page = identifier, filename = filename, index = 0,
319             last_column = len(data_rows[0]),
320             key_placement = key_placement, xlabel = xlabel, reps=reps,
321             miny=miny, grouptitle=grouptitle))
322
323
324 def print_benchmarks(data_headers, data_rows, title, group=None, variable=None,
325                     measure=None, convert_to_seconds=False, tinylabels=False,
326                     scriptlabels=False):
327     result = '#{0}\n'.format(title)
328     if group and variable and measure:
329         result = '#measure:{m} variable:{v} group:{g}'.format(

```

```

330         m=measure, v=variable, g=group)
331
332     prefix = ""
333     suffix = ""
334     if tinylabels:
335         prefix = "\\|\\|\\|tiny "
336     elif scriptlabels:
337         prefix = "\\|\\|\\|tiny{"
338         suffix = "}"
339     result = " ".join([format_value("{}{}{}".format(prefix, k, suffix))
340                        for k in data_headers])
341     result += '\\n'
342
343     for row in data_rows:
344         results = []
345         for i, v in enumerate(row):
346             convert = convert_to_seconds and i > 0
347             results.append(format_value(v, convert_to_seconds=convert))
348         result += ' '.join(results) + '\\n'
349     result += '\\n\\n'
350
351     return result
352
353 def format_value(value, convert_to_seconds=False):
354     if value == None:
355         return "-500"
356     if type(value) == str:
357         return "'{0}'".format(value)
358     if type(value) == int:
359         strval = str(value)
360         if convert_to_seconds == False:
361             return strval
362         strval = strval.zfill(10)
363         strlen = len(strval)
364         return "{}.{}".format(
365             strval[0:strlen-9],
366             strval[strlen-9:])
367
368     return str(value)
369
370 def hex_color_gradient(start, end, point):
371     # start, end are tuples with r,g,b values (integer)
372     # point is a point between 0 (start) and 1000 (end)
373     return "#" + "".join(
374         "{:0>2X}".format(
375             int(start[i] +
376                 ((end[i] - start[i]) * (float(point))))))
377         for i in range(0,3))

```

plot_data.py


```

1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 from collections import OrderedDict as odict
5 from itertools import groupby
6 from subprocess import call
7 from sys import argv
8 import functools
9 import pprint
10 import re
11 import os
12 import sys
13 import shutil
14 import uuid
15
16 import glob
17 import zipfile
18
19 import numpy
20 from numpy import array
21
22 from jni_types import primitive_type_definitions
23 from jni_types import object_type_definitions, array_types
24 from datafiles import read_datafiles, read_measurement_metadata
25 import analysis
26 from analysis import linear_fit, estimate_measuring_overhead
27 import gnuplot
28 import textualtable
29
30 FNULL = None
31
32 primitive_types = [
33     t['java']
34     for t in primitive_type_definitions
35 ]
36
37 reference_types = [
38     t['java']
39     for t in array_types.itervalues()
40 ]
41
42 reference_types.extend([
43     t['java']
44     for t in object_type_definitions
45 ])
46
47 types = reference_types + primitive_types
48
49 plot_axes = {
50     'description': 'operaatioiden määrä',
51     'parameter_count': 'kutsuparametrien määrä',
52     'dynamic_size': 'kohteen koko',
53     'direction': 'kutsusuunta',

```

```

54     'id': 'nimi'
55 }
56 pp = pprint.PrettyPrinter(depth=10, indent=4)
57
58 debugdata = open('/tmp/debug.txt', 'w')
59
60 def format_direction(fr, to, latex):
61     if fr == 'J':
62         fr = 'Java'
63     if to == 'J':
64         to = 'Java'
65     if latex:
66         SEPARATOR = '$\\\\\\rightarrow$'
67     else:
68         SEPARATOR = ' > '
69     return "%s%s%s" % (fr, SEPARATOR, to)
70
71 DIRECTIONS = [('C', 'J'), ('J', 'C'), ('J', 'J'), ('C', 'C')]
72
73 def preprocess_benchmarks(benchmarks, global_values, latex=None):
74     # For allocating benchmarks, the repetition count for individual benchmarks
75     # come from the datafile. For non-allocating, it is a global value.
76     keys = set([key for b in benchmarks for key in b.keys()])
77     if 'repetitions' in keys:
78         benchmarks = [b for b in benchmarks if b['repetitions'] is not None]
79     for b in benchmarks:
80         add_derived_values(b, latex=latex)
81         add_global_values(b, global_values)
82     return benchmarks
83
84 def add_derived_values(benchmark, latex=None):
85     # migration - todo - remove
86     if benchmark.get('response_time_millis') != None:
87         benchmark['response_time'] = benchmark.get('response_time_millis')
88         benchmark['time_unit'] = 'milliseconds'
89         del benchmark['response_time_millis']
90     if benchmark.get('dynamic_size') == None:
91         benchmark['dynamic_variation'] = 0
92         benchmark['dynamic_size'] = 0
93     else:
94         benchmark['dynamic_variation'] = 1
95     if benchmark['no'] == -1:
96         # Custom benchmark, do some name mapping:
97         bid = benchmark['id']
98         rename = True
99         if bid == 'CopyUnicode':
100             bid = 'GetStringRegion'
101         elif bid == 'CopyUTF':
102             bid = 'GetStringRegionUTF'
103         elif bid == 'StringLength':
104             bid = 'GetStringLength'
105         elif bid == 'StringLengthUTF':
106             bid = 'GetStringUTFLength'

```

```

107     elif bid == 'ReadUnicode':
108         bid = 'ReadString'
109     elif bid == 'ReadUnicodeCritical':
110         bid = 'ReadStringCritical'
111     elif bid == 'ReadUTF':
112         bid = 'ReadStringUTF'
113     elif bid == 'ReadUtf':
114         bid = 'ReadStringUTF'
115     elif bid == 'ReadObjectArrayElement':
116         bid = 'GetObjectArrayElement'
117     elif bid == 'WriteObjectArrayElement':
118         bid = 'SetObjectArrayElement'
119     else:
120         rename = False
121     if rename:
122         benchmark['id'] = bid
123
124     single_type = None
125     if (benchmark.get('parameter_count') == 0):
126         single_type = 'any'
127     elif (benchmark.get('parameter_type_count') == 1):
128         for tp in types:
129             if benchmark.get('parameter_type_{t}_count'.format(t=tp)) != None:
130                 single_type = tp
131                 break
132     benchmark['direction'] = format_direction(
133         benchmark['from'], benchmark['to'], latex)
134     benchmark['single_type'] = single_type
135     if 'Nio' in benchmark['id']:
136         benchmark['nio'] = True
137     else:
138         benchmark['nio'] = False
139
140 def add_global_values(benchmark, global_values):
141     for key, val in global_values.iteritems():
142         if key not in benchmark or benchmark[key] == None:
143             benchmark[key] = val
144         elif key == 'multiplier' and benchmark[key] != None:
145             benchmark[key] *= val
146
147
148 def extract_data(benchmarks,
149                 group=None, variable=None, measure=None,
150                 min_series_length=2, sort=None, min_series_width=None):
151
152     # info == extra metadata not to be analyzed
153     info = ['no', 'from', 'to', 'lineno', 'start', 'end']
154
155     if 'class' in benchmarks[0]:
156         info.append('class')
157     if 'description' in benchmarks[0]:
158         info.append('description')
159     if re.match('parameter_type_.+count', variable):

```

```

160         info.append('parameter_count')
161     if variable != 'id':
162         info.append('id')
163
164     # note: all the benchmarks have the same keyset
165     all_keys = set(benchmarks[0].keys())
166
167     # the actual keys of interest must have the least weight in sorting
168     sort_last = [group, variable, measure] + info
169     controlled_variables = all_keys - set(sort_last)
170     sorted_keys = list(controlled_variables) + sort_last
171
172     sorted_benchmarks = sorted(
173         benchmarks,
174         cmp=functools.partial(comp_function, sorted_keys))
175
176     # 1. group benchmarks into a multi-dimensional list
177     #     with the following structure:
178     #     - compatible-measurements (controlled variables are equal)
179     #     - plots (list of individual data series ie. plots)
180     #     - multiple measurements ()
181     benchmarks = group_by_keys(sorted_benchmarks, controlled_variables)
182     for i, x in enumerate(benchmarks):
183         benchmarks[i] = group_by_keys(x, [group])
184         for j, y in enumerate(benchmarks[i]):
185             benchmarks[i][j] = group_by_keys(y, [variable])
186
187     # 2. statistically combine multiple measurements
188     # for the exact same benchmark and parameters,
189     # and store information about the roles of keys
190
191     for i, compatibles in enumerate(benchmarks):
192         for j, plotgroups in enumerate(compatibles):
193             for k, measured_values in enumerate(plotgroups):
194
195                 plotgroups[k] = aggregate_measurements(
196                     measured_values, measure, stat_fun=min)
197
198                 compatibles[j] = odict(
199                     (benchmark[variable], {
200                         'fixed': dict(
201                             (key, benchmark[key]) for key in controlled_variables),
202                         'info': dict((key, benchmark[key]) for key in info),
203                         'variable': variable,
204                         'measure': measure,
205                         'group': group,
206                         variable: benchmark[variable],
207                         measure: benchmark[measure],
208                         group: benchmark[group]
209                     }) for benchmark in plotgroups)
210
211     benchmarks[i] = odict(
212         sorted(((bms.values()[0][group], bms)

```

```

213         for bms in benchmarks[i]),
214         key=lambda x: x[0]))
215
216     return [x for x in benchmarks
217             if len((x.values())[0]) >= min_series_length]
218
219
220 def group_by_keys(sorted_benchmarks, keyset):
221     # todo make into generator?
222     return [
223         list(y) for x, y in groupby(
224             sorted_benchmarks,
225             key=lambda b: [b[k] for k in keyset])]
226
227
228 def aggregate_measurements(benchmarks, measure, stat_fun=min):
229     values = []
230     benchmark = None
231     for benchmark in benchmarks:
232         values.append(benchmark[measure])
233
234     benchmark[measure] = stat_fun(values)
235
236     if len(values) != benchmark['multiplier']:
237         print ("Error: expecting", benchmark['multiplier'],
238              "measurements, got", len(values))
239         debugdata.write(pp.pformat(list(benchmarks)))
240         exit(1)
241
242     return benchmark
243
244
245 def comp_function(keys, left, right):
246     for key in keys:
247         if key not in left and key not in right:
248             continue
249         l, r = left[key], right[key]
250         if l < r:
251             return -1
252         if l > r:
253             return 1
254     return 0
255
256
257 def without(keys, d):
258     if keys == None:
259         return d
260     return dict(((key, val) for key, val in d.iteritems() if key not in keys))
261
262
263 def plot(
264     benchmarks, gnuplot_script, plotpath, metadata_file,
265     keys_to_remove=None, select_predicate=None,

```

```

266         group=None, variable=None, measure=None,
267         title=None, style=None, min_series_width=1,
268         key_placement='inside top left',
269         identifier=None,
270         revision=None, checksum=None, output='pdf'):
271
272     if len(benchmarks) > 0 and benchmarks[0].get('is_allocating'):
273         identifier += '-alloc'
274     if len(benchmarks) > 0:
275         reps = benchmarks[0].get('repetitions')
276
277     filtered_benchmarks = [
278         without(keys_to_remove, x)
279         for x in benchmarks
280         if select_predicate(x)]
281
282     variables = set([benchmark[variable] for benchmark in filtered_benchmarks])
283
284     if len(variables) < 2:
285         print 'Skipping plot without enough data variables', title
286         return
287
288     if len(filtered_benchmarks) == 0:
289         print 'Error, no benchmarks for', title
290         exit(1)
291
292     print 'Plotting', title
293
294     specs = {
295         'group': group,
296         'variable': variable,
297         'measure': measure}
298
299     data = extract_data(filtered_benchmarks, **specs)
300
301     index = -1
302
303     data_len = len([s for s in data if len(s.keys()) >= min_series_width])
304     for series in data:
305         if len(series.keys()) < min_series_width:
306             # there are not enough groups to display
307             continue
308         index += 1
309
310         plot.page += 1
311         axes_label = plot_axes.get(variable, '<unknown variable>')
312
313         headers, rows = make_table(
314             series, group, variable, measure, axes_label)
315
316         assert identifier is not None
317         id_suffix = ""
318         if data_len > 1:

```

```

319         id_suffix = "-{}".format(index)
320
321     gnuplot.output_plot(
322         headers, rows, plotpath, gnuplot_script,
323         title, specs, style, plot.page, identifier + id_suffix, axes_label,
324         output=output, key_placement=key_placement, reps=reps
325     )
326
327     metadata_file.write("\n\n{0}\n{1}\n\n".format(
328         title, identifier + id_suffix))
329
330     keyvalpairs = series.values()[0].values()[0]['fixed'].items() + [
331         ('variable', axes_label),
332         ('measure', measure),
333         ('grouping', group)]
334
335     for k, v in keyvalpairs:
336         if v != None:
337             metadata_file.write("{k:<25} {v}\n".format(k=k, v=v))
338
339     metadata_file.write(
340         "\n" + textualtable.make_textual_table(headers, rows))
341
342     id_headers, id_rows = make_table(
343         series, group, variable, 'class', axes_label)
344
345     def make_id(variable_value, item, variable):
346         ret = "/".join([revision, item or '-'])
347         if variable == 'dynamic_size':
348             ret += "/" + str(variable_value)
349         return ret
350
351     id_rows = [
352         [row[0]] +
353         [make_id(row[0], item, variable) for item in row[1:]]
354         for row in id_rows]
355
356     ttable = textualtable.make_textual_table(id_headers, id_rows)
357     metadata_file.write("\n" + ttable)
358
359     if variable != 'direction' and variable != 'id':
360         x, polys, residuals = linear_fit(rows)
361
362         fitted_curves = []
363         for i, xval in enumerate(x):
364             current = [xval]
365             current.extend(rows[i][1:])
366             current.extend([numpy.polyval(polys[j], xval)
367                             for j in range(0, len(rows[i]) - 1)])
368             fitted_curves.append(current)
369
370     plot.page += 1
371     gnuplot.output_plot(

```

```

372         headers + headers[1:], fitted_curves, plotpath, gnuplot_script,
373         title, specs, 'fitted_lines', plot.page, identifier +
374         id_suffix + '-fit', axes_label, output=output, reps=reps)
375
376     def simplified_function(poly):
377         return "{:.3g} * x {:.3g}".format(poly[0], poly[1])
378     metadata_file.write(
379         "\npolynomial:\n" + textualltable.make_vertical_textual_table(
380             headers[1:], [map(simplified_function, polys)]))
381     metadata_file.write(
382         "\nresiduals:\n" + textualltable.make_vertical_textual_table(
383             headers[1:], [residuals]))
384     metadata_file.write(
385         "\nslope:\n" + textualltable.make_vertical_textual_table(
386             headers[1:], [map(lambda p: p[0], polys)]))
387     metadata_file.write(
388         "\nintercept:\n" + textualltable.make_vertical_textual_table(
389             headers[1:], [map(lambda p: p[1], polys)]))
390     return data
391
392 plot.page = 0
393
394 def convert_to_seconds(value):
395     if type(value) == int:
396         strval = str(value)
397         if convert_to_seconds == False:
398             return strval
399         strval = strval.zfill(10)
400         strlen = len(strval)
401         return float("{}.{ {}".format(
402             strval[0:strlen-9],
403             strval[strlen-9:]))
404     return value
405
406 def make_table(series, group, variable, measure, axes_label):
407     all_benchmark_variables_set = set()
408     for bm_list in series.itervalues():
409         all_benchmark_variables_set.update(bm_list.keys())
410
411     all_benchmark_variables = sorted(list(all_benchmark_variables_set))
412
413     rows = []
414
415     headers = (
416         [axes_label] +
417         [k for k in series.iterkeys()]
418     )
419
420     for v in all_benchmark_variables:
421         row = []
422         row.append(v)
423         for key, grp in series.iteritems():
424             val = grp.get(v, {}).get(measure, None)

```



```

425         if val is None:
426             val = grp.get(v, {}).get('info', {}).get(measure, None)
427         if measure == 'response_time':
428             val = convert_to_seconds(val)
429         row.append(val)
430     rows.append(row)
431
432     if variable == 'id':
433         rows = sorted(rows, key=lambda x: x[1] or -1)
434
435     return headers, rows
436
437
438 def binned_value(minimum, width, value):
439     return width * (int(value - minimum) / int(width)) + minimum
440
441
442 def plot_distributions(
443     all_benchmarks, output, plotpath, gnuplotcommands, bid,
444     metadata_file, plot_type=None, latex=None, **kwargs):
445
446     output_type = 'screen'
447     if plot_type != 'animate':
448         output_type = 'pdf'
449
450     gnuplot.init(gnuplotcommands, output, bid, output_type=output_type)
451     measure = 'response_time'
452
453     keyset = set(all_benchmarks[0].keys()) - \
454         set([measure, 'lineno', 'start', 'end'])
455     comparison_function = functools.partial(comp_function, keyset)
456     sorted_benchmarks = sorted(all_benchmarks, cmp=comparison_function)
457
458     for group in group_by_keys(sorted_benchmarks, keyset):
459         if plot_type != None:
460             keyf = lambda x: x['lineno']
461         else:
462             keyf = lambda x: x[measure]
463
464         frame_count = 1
465         if plot_type != None:
466             frame_count = 256
467
468         current_frame = frame_count
469         all_values = [b[measure] for b in sorted(group, key=keyf)]
470         while current_frame > 0:
471
472             if current_frame == frame_count:
473                 frame_ratio = 1
474             else:
475                 frame_ratio = float(current_frame) / frame_count
476             values = array(all_values[0:int(frame_ratio * len(all_values))])
477

```

```

478     bin_width = 500
479     min_x = numpy.amin(all_values)
480     max_x = numpy.amax(all_values)
481
482     bin_no = (max_x - min_x) / bin_width
483
484     hgram, bin_edges = numpy.histogram(values, bins=max(bin_no, 10))
485
486     mode = bin_edges[numpy.argmax(hgram)]
487     min_x = mode - 100000
488     max_x = mode + 100000
489
490     if current_frame == frame_count:
491         metadata_file.write(
492             'Direction {0}\n'.format(group[0]['direction']))
493
494         gnuplotcommands.write(
495             gnuplot.templates['binned_init'].format(
496                 title='%s %s' % (group[0]['id'], group[
497                     0]['direction']),
498                 binwidth=bin_edges[1] - bin_edges[0],
499                 min_x=min_x, max_x=max_x,
500                 max_y=numpy.max(hgram)))
501
502         if plot_type == 'animate':
503             gnuplotcommands.write('pause -1\n')
504
505         elif plot_type == 'gradient':
506             gnuplotcommands.write("set multiplot\n")
507
508     current_frame -= 1
509
510     if plot_type == None:
511         gnuplotcommands.write(
512             gnuplot.templates['binned_frame'].format(
513                 datapoints='', color='#000033',
514                 values='\n'.join(['{} {} {}'.format(val, count, val)
515                                     for val, count in zip(
516                                         bin_edges, hgram)])))
517
518     elif plot_type == 'gradient':
519         gnuplotcommands.write(
520             gnuplot.templates['binned_frame'].format(
521                 datapoints='',
522                 color=gnuplot.hex_color_gradient(
523                     (125, 0, 0), (255, 255, 0), 1 - frame_ratio),
524                 values='\n'.join(['{} {} {}'.format(val, count, val)
525                                     for val, count in zip(
526                                         bin_edges, hgram)])))
527
528     gnuplotcommands.write("set xtics\n")
529     gnuplotcommands.write("set ytics\n")
530

```

```

531
532 def plot_benchmarks(
533     all_benchmarks, output, plotpath, gnuplotcommands, bid, metadata_file,
534     plot_type=None, revision=None, checksum=None, latex=None):
535
536     output_type = 'pdf'
537     if latex == 'plotlatex':
538         output_type = 'latex'
539     elif latex == 'plotsvg':
540         output_type = 'svg'
541
542     gnuplot.init(gnuplotcommands, output, bid, output_type=output_type)
543
544     type_counts = ["parameter_type_{t}_count".format(t=tp) for tp in types]
545     keys_to_remove = type_counts[:]
546     keys_to_remove.extend(
547         ['parameter_type_count', 'single_type', 'dynamic_variation'])
548
549     benchmarks = [bm for bm in all_benchmarks if bm['no'] != -1]
550     defaults = [benchmarks, gnuplotcommands, plotpath]
551
552 #     analysis.calculate_overheads()
553     overhead_estimates = {}
554     overhead_benchmarks = [
555         bm for bm in all_benchmarks
556         if bm['no'] == -1 and 'Overhead' in bm['id']]
557     for loop_type in ['AllocOverhead', 'NormalOverhead']:
558         for from_lang in ['C', 'J']:
559             language_name = from_lang
560             if language_name == 'J': language_name = 'Java'
561             overhead_estimates[from_lang] = {}
562             overhead_data = plot(
563                 overhead_benchmarks, gnuplotcommands, plotpath, metadata_file,
564                 style='simple_groups',
565                 key_placement=None,
566                 title='Mittauksen perusrasite ({}).format(language_name),
567                 identifier='{}-{}'.format(loop_type.lower(), from_lang.lower()),
568                 keys_to_remove=[],
569                 select_predicate=(
570                     lambda x: x['from'] == from_lang and loop_type in x['id'])
571                 group='from',
572                 measure='response_time',
573                 variable='description',
574                 revision=revision,
575                 checksum=checksum,
576                 output=output_type)
577
578             if overhead_data == None:
579                 continue
580             if len(overhead_data) > 1:
581                 print('Error, more loop types than expected.',
582                     len(overhead_data))
583                 exit(1)

```

```

584
585     series = overhead_data[0]
586     headers, rows = make_table(series,
587                                'from',
588                                'description',
589                                'response_time',
590                                'workload')
591     est = estimate_measuring_overhead(rows[1:])
592     overhead_estimates[from_lang][loop_type] = est[0]
593     metadata_file.write('Overhead ' + from_lang + ' ' + str(est[0]))
594
595     for i, ptype in enumerate(types):
596         plot(
597             benchmarks, gnuplotcommands, plotpath, metadata_file,
598             title='{}-tyyppiset kutsuparametrit'.format(ptype),
599             identifier='basic-call-{}'.format(ptype),
600             style='simple_groups',
601             keys_to_remove=(
602                 keys_to_remove +
603                 ['dynamic_size'] +
604                 ['has_reference_types']),
605             select_predicate=lambda x: (
606                 x['single_type'] in [ptype, 'any'] and
607                 x['dynamic_size'] == 0),
608             group='direction',
609             variable='parameter_count',
610             measure='response_time',
611             revision=revision, checksum=checksum, output=output_type)
612
613     for fr, to in DIRECTIONS:
614         direction = format_direction(fr, to, latex)
615         plot(
616             benchmarks, gnuplotcommands, plotpath, metadata_file,
617             title='Vaihteleva argumentin koko kutsusuunnassa ' + direction,
618             identifier='variable-argument-size-{}-{}'.format(fr.lower(),
619                                                                to.lower()),
620             style='simple_groups',
621             keys_to_remove=type_counts,
622             select_predicate=(
623                 lambda x: (
624                     x['direction'] == direction and
625                     x['has_reference_types'] == 1 and
626                     x['single_type'] in reference_types and
627                     x['parameter_count'] == 1)),
628             group='single_type',
629             variable='dynamic_size',
630             measure='response_time',
631             revision=revision, checksum=checksum, output=output_type)
632
633     for fr, to in DIRECTIONS:
634         direction = format_direction(fr, to, latex)
635         plot(
636             benchmarks, gnuplotcommands, plotpath, metadata_file,

```

```

637         title='Vaihteleva paluuarvon koko kutsusuunnassa ' + direction,
638         identifier='variable-return-value-size-{}-{}'.format(fr.lower(),
639                                                             to.lower()),
640         style='simple_groups',
641         keys_to_remove=type_counts,
642         select_predicate=(
643             lambda x: x['has_reference_types'] == 1
644             and x['direction'] == direction
645             and x['return_type'] != 'void'),
646         group='return_type',
647         variable='dynamic_size',
648         measure='response_time',
649         revision=revision, checksum=checksum, output=output_type)
650
651     keys_to_remove = type_counts[:]
652     keys_to_remove.append('has_reference_types')
653     keys_to_remove.append('dynamic_variation')
654
655     for fr, to in DIRECTIONS:
656         direction = format_direction(fr, to, latex)
657         plot(
658             benchmarks, gnuplotcommands, plotpath, metadata_file,
659             style='simple_groups',
660             title='Parametrityyppien vertailu ' + direction,
661             identifier='basic-call-all-types-{}-{}'.format(fr.lower(),
662                                                             to.lower()),
663             keys_to_remove=keys_to_remove,
664             select_predicate=(
665                 lambda x: x['direction'] == direction),
666             group='single_type',
667             variable='parameter_count',
668             measure='response_time',
669             revision=revision, checksum=checksum, output=output_type)
670
671     plot(
672         benchmarks, gnuplotcommands, plotpath, metadata_file,
673         style='named_columns',
674         title='Paluuarvon tyyppit',
675         identifier='return-value-types',
676         keys_to_remove=['has_reference_types', 'dynamic_variation'],
677         select_predicate=(
678             lambda x: x['dynamic_size'] == 0 and
679             x['return_type'] != 'void'),
680         group='return_type',
681         measure='response_time',
682         variable='direction',
683         min_series_width=2,
684         revision=revision, checksum=checksum, output=output_type)
685     # had: sort 'response_time', min_series_width: 2 , unused?
686
687     def utf(b):
688         return 'UTF' in b['id'] or 'Utf' in b['id']
689

```

```

690 filters = {
691     'utf': utf,
692     'arrayregion': lambda x: 'ArrayRegion' in x['id'],
693     'bytebufferview': lambda x: 'ByteBufferView' in x['id'],
694     'unicode': lambda b: not utf(b) and 'String' in b['id'],
695     'arrayelements': (lambda x:
696         'ArrayElements' in x['id'] or
697         'ArrayLength' in x['id'] or
698         'ReadPrimitive' in x['id']),
699 }
700 def uncategorized(x):
701     for f in filters.values():
702         if f(x):
703             return False
704     return True
705
706 benchmarks = {}
707 for key, f in filters.iteritems():
708     benchmarks[key] = [
709         bm for bm in all_benchmarks
710         if bm['no'] == -1 and f(bm)]
711
712 benchmarks['uncategorized'] = [
713     bm for bm in all_benchmarks
714     if bm['no'] == -1 and 'Overhead' not in bm['id'] and uncategorized(bm)]
715
716 custom_benchmarks = benchmarks['uncategorized']
717
718 for fr, to in DIRECTIONS:
719     direction = format_direction(fr, to, latex)
720     plot(
721         custom_benchmarks, gnuplotcommands, plotpath, metadata_file,
722         style='simple_groups',
723         title='Erityiskutsut suunnassa ' + direction,
724         identifier='special-calls-{}-{}'.format(fr.lower(), to.lower()),
725         select_predicate=(
726             lambda x: (x['direction'] == direction and
727                 x['dynamic_variation'] == 1)),
728         group='id',
729         measure='response_time',
730         variable='dynamic_size',
731         revision=revision, checksum=checksum, output=output_type)
732
733     plot(
734         benchmarks['arrayregion'], gnuplotcommands, plotpath,
735         metadata_file,
736         style='simple_groups',
737         title='Erityiskutsut suunnassa ' + direction,
738         identifier='special-calls-arrayregion-{}-{}'.format(fr.lower(),
739             to.lower()),
740         select_predicate=(
741             lambda x: (x['direction'] == direction and
742                 x['dynamic_variation'] == 1)),

```

```

743         group='id',
744         measure='response_time',
745         variable='dynamic_size',
746         revision=revision, checksum=checksum, output=output_type)
747
748     plot(
749         benchmarks['arrayelements'], gnuplotcommands, plotpath,
750         metadata_file,
751         style='simple_groups',
752         title='Erityiskutsut suunnassa ' + direction,
753         identifier='special-calls-arrayelements-{}-{}'.format(fr.lower(),
754                                                                to.lower()),
755         select_predicate=(
756             lambda x: (x['direction'] == direction and
757                        x['dynamic_variation'] == 1)),
758         group='id',
759         measure='response_time',
760         variable='dynamic_size',
761         revision=revision, checksum=checksum, output=output_type)
762
763     plot(
764         benchmarks['utf'], gnuplotcommands, plotpath, metadata_file,
765         style='simple_groups',
766         title='UTF-merkkijonot suunnassa ' + direction,
767         identifier='special-calls-utf-{}-{}'.format(fr.lower(),
768                                                    to.lower()),
769         select_predicate=(
770             lambda x: (x['direction'] == direction and
771                        x['dynamic_variation'] == 1)),
772         group='id',
773         measure='response_time',
774         variable='dynamic_size',
775         revision=revision, checksum=checksum, output=output_type)
776
777     plot(
778         benchmarks['unicode'], gnuplotcommands, plotpath, metadata_file,
779         style='simple_groups',
780         key_placement='inside bottom left',
781         title='Unicode-merkkijonot suunnassa ' + direction,
782         identifier='special-calls-unicode-{}-{}'.format(fr.lower(),
783                                                         to.lower()),
784         select_predicate=(
785             lambda x: (x['direction'] == direction and
786                        x['dynamic_variation'] == 1)),
787         group='id',
788         measure='response_time',
789         variable='dynamic_size',
790         revision=revision, checksum=checksum, output=output_type)
791
792     plot(
793         benchmarks['bytebufferview'], gnuplotcommands, plotpath,
794         metadata_file,
795         style='simple_groups',

```

```

796     title='Erityiskutsut suunnassa ' + direction,
797     identifier='special-calls-bytebufferview-{}-{}'.format(fr.lower(),
798                                                             to.lower()),
799     select_predicate=(
800         lambda x: (x['direction'] == direction and
801                    x['dynamic_variation'] == 1 and
802                    'Bulk' not in x['id'])),
803     group='id',
804     measure='response_time',
805     variable='dynamic_size',
806     revision=revision, checksum=checksum, output=output_type)
807
808     plot(
809         benchmarks['bytebufferview'], gnuplotcommands, plotpath,
810         metadata_file,
811         style='simple_groups',
812         title='Erityiskutsut suunnassa ' + direction,
813         identifier='special-calls-bulk-bytebufferview-{}-{}'.format(
814             fr.lower(), to.lower()),
815         select_predicate=(
816             lambda x: (x['direction'] == direction and
817                       x['dynamic_variation'] == 1 and
818                       'Bulk' in x['id'])),
819         group='id',
820         measure='response_time',
821         variable='dynamic_size',
822         revision=revision, checksum=checksum, output=output_type)
823
824     plot(
825         custom_benchmarks, gnuplotcommands, plotpath, metadata_file,
826         style='histogram',
827         title='Erityiskutsujen vertailu eri kutsusuunnissa',
828         identifier='special-calls-non-dynamic',
829         select_predicate=(
830             lambda x: (
831                 x['dynamic_variation'] == 0 and
832                 'Field' in x['id'])),
833         group='direction',
834         measure='response_time',
835         variable='id',
836         revision=revision, checksum=checksum, output=output_type)
837
838
839     MEASUREMENT_FILE = 'measurements.txt'
840     DEVICE_PATH = '/sdcard/results'
841     PLOTPATH = '/tmp'
842     TOOL_NAMESPACE = 'fi.helsinki.cs.tituomin.nativebenchmark.measuringtool'
843
844
845     def sync_measurements(dev_path, host_path, filename, update=True):
846         old_path = host_path + '/' + filename
847         tmp_path = '/tmp/' + filename
848         if not update and os.path.exists(old_path):

```



```

849     print 'No sync necessary'
850     return
851
852     kwargs = {}
853     if FNULL is not None:
854         kwargs['stdout'] = FNULL
855         kwargs['stderr'] = FNULL
856
857     try:
858         success = call(['adb', 'pull',
859                        dev_path + '/' + filename,
860                        tmp_path], **kwargs)
861     except OSError:
862         success = -1
863     if success == 0:
864         if os.path.exists(old_path):
865             size_new = os.path.getsize(tmp_path)
866             size_old = os.path.getsize(old_path)
867             if size_new < size_old:
868                 print ("Warning: new file contains less data than "
869                       "the old. Aborting.")
870                 exit(2)
871             shutil.move(tmp_path, old_path)
872
873     else:
874         print "Could not get new measurements, continuing with old."
875
876 def render_perf_reports_for_measurement(identifier, measurements,
877                                       measurement_path, output_path,
878                                       output_command=False):
879     path = identifier.split("/")
880     if len(path) < 2:
881         print 'Invalid identifier {}'.format(identifier)
882         exit(1)
883     if len(path) == 3:
884         revision, class_, dynamic_size = path
885     elif len(path) == 2:
886         revision, class_ = path
887         dynamic_size = None
888
889     def match_measurement(measurement):
890         m = measurement[0]
891         return (m.get('code-revision') == revision and
892                m.get('tool') == 'LinuxPerfRecordTool')
893
894     def match_measurement_run(m):
895         if m.get('class').lower() != class_.lower():
896             return False
897         if dynamic_size and m.get('dynamic_size') != int(dynamic_size):
898             return False
899         if 'Filename' not in m or m['Filename'] is None:
900             return False
901         return True

```

```

902
903 datafiles = []
904 for measurement in filter(match_measurement, measurements):
905     mid = measurement[0].get('id')
906     zpath = os.path.join(measurement_path, 'perfdata-{}.zip'.format(mid))
907     try:
908         measurement_zipfile = zipfile.ZipFile(zpath, 'r')
909         datafiles.append({
910             'zip': measurement_zipfile,
911             'zip_path': zpath,
912             'mid': mid,
913             'csv': measurement_zipfile.open(
914                 '{}/benchmarks-{}.csv'.format(mid))
915         })
916     except zipfile.BadZipfile:
917         print 'Bad zip file %s' % zpath
918     except IOError as e:
919         print 'Problem with zip file %s' % zpath
920         print e
921
922 benchmarks = []
923 for df in datafiles:
924     benchmarks.append({
925         'zip': df['zip'],
926         'mid': df['mid'],
927         'metadata': read_datafiles([df['csv']], silent=output_command)
928     })
929
930 matching_benchmarks = []
931 for bm in benchmarks:
932     for row in bm['metadata']:
933         if match_measurement_run(row):
934             matching_benchmarks.append({
935                 'zip': bm['zip'],
936                 'mid': bm['mid'],
937                 'filename': row['Filename']
938             })
939
940 for record in matching_benchmarks:
941     perf_file = record['zip'].extract('{}{}'.format(record['mid'],
942                                                     record['filename']),
943                                       '/tmp')
944     try:
945         command_parts = [
946             "perf report",
947             "-i {}",
948             "--header",
949             "--syms=/home/tituomin/droid-symbols",
950             "--kallsyms=/home/tituomin/droid/linux-kernel/kallsyms"
951         ]
952         command_parts.extend([
953             "-g graph,0,caller",
954             "--stdio",

```

```

955         "| c++filt",
956         ">/tmp/out.txt"
957     ])
958     command = " ".join(command_parts).format(perf_file)
959     if output_command:
960         print command
961         exit(0)
962     else:
963         call([command], shell=True)
964     except OSError as e:
965         print e.filename, e.message, e.args
966
967     for f in datafiles:
968         f['zip'].close()
969     print "Profile for identifier", identifier
970     with open('/tmp/out.txt', 'r') as f:
971         print f.read()
972     exit(0)
973
974 if __name__ == '__main__':
975     if len(argv) < 4 or len(argv) > 6:
976         print argv[0]
977         print ("\n    Usage: %s input_path output_path "
978               "limit [pdfviewer] [separate]\n").format(argv[0])
979         exit(1)
980
981     FNULL = open(os.devnull, 'w')
982
983     method = argv[0]
984     measurement_path = os.path.normpath(argv[1])
985     output_path = argv[2]
986
987     if 'plotlatex' in method:
988         latex = 'plotlatex'
989         method = 'curves'
990     elif 'plotsvg' in method:
991         latex = 'plotsvg'
992         method = 'curves'
993     else:
994         latex = None
995
996     output_command = False
997     if len(argv) > 5:
998         if argv[5] == 'show-command':
999             output_command = True
1000
1001     limit = argv[3]
1002     if len(argv) > 4:
1003         pdfviewer = argv[4]
1004     else:
1005         pdfviewer = None
1006
1007     if len(argv) == 6:

```

```

1008     group = (not argv[5] == "separate")
1009 else:
1010     group = True
1011
1012 if output_command:
1013     system_stdout = sys.stdout
1014     system_stderr = sys.stderr
1015     sys.stdout = FNULL
1016     sys.stderr = FNULL
1017
1018 sync_measurements(DEVICE_PATH, measurement_path, MEASUREMENT_FILE)
1019
1020 f = open(os.path.join(measurement_path, MEASUREMENT_FILE))
1021
1022 try:
1023     measurements = read_measurement_metadata(f, group)
1024 finally:
1025     f.close()
1026
1027 limited_measurements = (
1028     filter(lambda x: int(x[0].get('repetitions', 0)) >= int(limit),
1029           measurements.values()))
1030
1031 # ID = revision/checksum/class[/dynamic_size]
1032 if 'perf_select' in method:
1033     identifier = argv[4]
1034     if output_command:
1035         sys.stdout = system_stdout
1036         sys.stderr = system_stderr
1037         FNULL.close()
1038     render_perf_reports_for_measurement(
1039         identifier, limited_measurements, measurement_path,
1040         output_path, output_command=output_command)
1041     exit(0)
1042
1043 csv_files = set()
1044 for f in glob.iglob(measurement_path + '/benchmarks-*.csv'):
1045     try:
1046         csv_files.add(f.split('.')[0].split('benchmarks-')[1])
1047     except IndexError:
1048         pass
1049
1050 if len(limited_measurements) > 20:
1051     i = len(limited_measurements) - 20 + 1
1052     splice = limited_measurements[-20:]
1053 else:
1054     i = 1
1055     splice = limited_measurements
1056
1057 print "\nAvailable compatible measurements. Choose one"
1058 for m in splice:
1059     b = m[0]
1060     warning = ""

```

```

1061         if int(b.get('rounds')) == 0:
1062             warning = " <---- WARNING INCOMPLETE MEASUREMENT"
1063         print """
1064 [{idx}]:      total measurements: {num}
1065                  local: {local}
1066                  repetitions: {reps}
1067                  description: {desc}
1068                  rounds: {rounds}{warning}
1069                  id: {mid}
1070                  checksum: {ck}
1071                  revision: {rev}
1072                  tool: {tool}
1073                  cpu: {freq} KHz
1074                  set: {bset}
1075                  filter: {sfilter}
1076                  dates: {first} -
1077                        {last}
1078 """.format(
1079     local=b.get('id') in csv_files,
1080     num=len(m),
1081     mid=b.get('id'),
1082     idx=i,
1083     warning=warning,
1084     last=m[-1]['end'],
1085     rounds=reduce(lambda x, y: y + x, [int(b['rounds']) for b in m]),
1086     reps=b.get('repetitions'),
1087     ck=b.get('code-checksum'),
1088     rev=b.get('code-revision'),
1089     tool=b.get('tool'),
1090     freq=b.get('cpu-freq'),
1091     bset=b.get('benchmark-set'),
1092     desc=b.get('description'),
1093     sfilter=b.get('substring-filter'),
1094     first=b.get('start')
1095 )
1096
1097     i += 1
1098
1099 try:
1100     response = raw_input("Choose set 1-{last} >> ".format(last=i - 1))
1101 except EOFError:
1102     print 'Exiting.'
1103     exit(1)
1104
1105 benchmark_group = limited_measurements[int(response) - 1]
1106
1107 filenames = []
1108 ids = []
1109 multiplier = 0
1110 for measurement in benchmark_group:
1111     if 'LinuxPerfRecordTool' in measurement['tool']:
1112         basename = "perfddata-{n}.zip"
1113     else:

```

```

1114         basename = "benchmarks-{n}.csv"
1115     filenames.append(
1116         basename.format(n=measurement['id']))
1117     if 'logfile' in measurement:
1118         filenames.append(measurement['logfile'])
1119     ids.append(measurement['id'])
1120     multiplier += int(measurement['rounds'])
1121
1122     files = []
1123     for filename in filenames:
1124         sync_measurements(DEVICE_PATH, measurement_path,
1125                             filename, update=False)
1126         if filename not in [m.get('logfile') for m in benchmark_group]:
1127             files.append(open(os.path.join(measurement_path, filename)))
1128
1129     first_measurement = benchmark_group[0]
1130
1131     global_values = {
1132         'repetitions': first_measurement['repetitions'],
1133         'is_allocating': first_measurement['benchmark-set'] == 'ALLOC',
1134         'multiplier': multiplier
1135     }
1136
1137     perf = False
1138     if 'LinuxPerfRecordTool' in first_measurement['tool']:
1139         print 'Perf data downloaded.'
1140         perf = True
1141     if not perf:
1142         try:
1143             benchmarks = read_datafiles(files)
1144
1145         finally:
1146             for f in files:
1147                 f.close()
1148
1149     benchmark_group_id = os.getenv('PLOT_ID', str(uuid.uuid4()))
1150     plot_prefix = 'plot-{0}'.format(benchmark_group_id)
1151
1152     if latex is not None:
1153         output_filename = os.path.join(output_path, plot_prefix)
1154     else:
1155         output_filename = os.path.join(output_path, plot_prefix + '.pdf')
1156     plot_filename = plot_prefix + '.gp'
1157
1158     plotfile = open(os.path.join(output_path, plot_filename), 'w')
1159     metadata_file = open(os.path.join(
1160         output_path, plot_prefix + '-metadata.txt'), 'w')
1161
1162     measurement_ids = " ".join(ids)
1163     metadata_file.write("-- mode: perf-report; --\n\n")
1164     metadata_file.write("id: {0}\n".format(benchmark_group_id))
1165     metadata_file.write("measurements: {0}\n".format(measurement_ids))
1166

```

```

1167     benchmarks = preprocess_benchmarks(benchmarks, global_values,
1168                                         latex=latex)
1169
1170     animate = False
1171     if pdfviewer == 'anim':
1172         plot_type = 'animate'
1173         pdfviewer = None
1174     elif pdfviewer == 'gradient':
1175         plot_type = 'gradient'
1176         pdfviewer = None
1177     else:
1178         plot_type = None
1179
1180     if 'curves' in method:
1181         function = plot_benchmarks
1182     elif 'distributions' in method:
1183         function = plot_distributions
1184     if perf or not function:
1185         exit(0)
1186
1187     function(
1188         benchmarks,
1189         output_filename,
1190         PLOTPATH,
1191         plotfile,
1192         benchmark_group_id,
1193         metadata_file,
1194         plot_type=plot_type,
1195         revision=first_measurement['code-revision'],
1196         checksum=first_measurement['code-checksum'],
1197         latex=latex)
1198
1199     plotfile.flush()
1200     plotfile.close()
1201     if plot_type == 'animate':
1202         print "Press enter to start animation."
1203         call(["gnuplot", plotfile.name])
1204     if pdfviewer:
1205         call([pdfviewer, str(output_filename)])
1206     print "Final plot",
1207     if 'animate' != plot_type:
1208         print str(output_filename)
1209     else:
1210         print str(plot_filename)
1211     print(benchmark_group_id)
1212     exit(0)

```

textualtable.py

```

1 #/usr/bin/python
2
3 def make_textual_table(headers, rows):
4     result = ""
5     max_widths = []
6
7     for x in headers:
8         max_widths.append(len(str(x)))
9
10    for row in rows:
11        for i, x in enumerate(row):
12            l = len(str(x))
13            if max_widths[i] < l:
14                max_widths[i] = l
15
16    row_format = ["{:{:>{w}}}" for w in max_widths]
17    row_format = "".join(row_format) + "\n"
18
19    result += row_format.format(*headers)
20    for row in rows:
21        result += row_format.format(*row)
22    return result
23
24 def make_vertical_textual_table(headers, elements):
25     result = ""
26     max_width = max((len(x) for x in headers))
27
28     header_format = "{:{:>{w}}}" .format(w=max_width)
29
30     for i in range(0, len(headers)):
31         result += header_format.format(headers[i])
32         for group in elements:
33             result += "    "
34             result += str(group[i])
35         result += "\n"
36
37    return result

```