

# Capture-Replay Tests in J2ME

*Testy capture-replay w środowisku J2ME*

Marcin Zduniak   Bartosz Walter   Dawid Weiss

Institute of Computing Science  
Poznan University of Technology

2007



## Who is who

- Marcin Zduniak (the graduate)
- Bartosz Walter (thesis supervisor)
- Dawid Weiss (original concept, mentoring)

# What are “software tests”?

## What are “software tests”?

(Wikipedia)

**Software testing** is the process used to help identify the **correctness**, **completeness**, **security**, and **quality** of developed computer software.

How can we “test software”?

## Unit tests

Correctness of individual units of source code

## Module/ integration tests

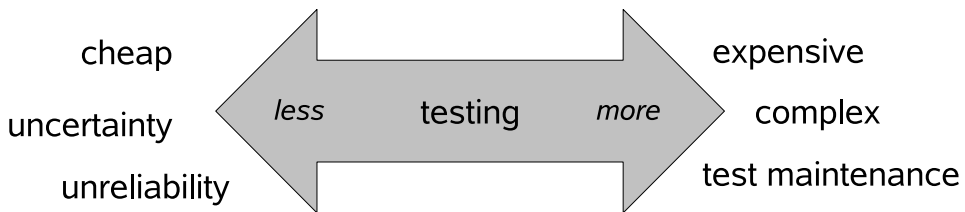
Chunks of functionality, sometimes the entire program. testing in various target environments (O/S's, processors etc.).

## Acceptance tests

Compliance to customer's requirements; often manual work.

## Regression tests

System stability/ correctness in response to ongoing changes.

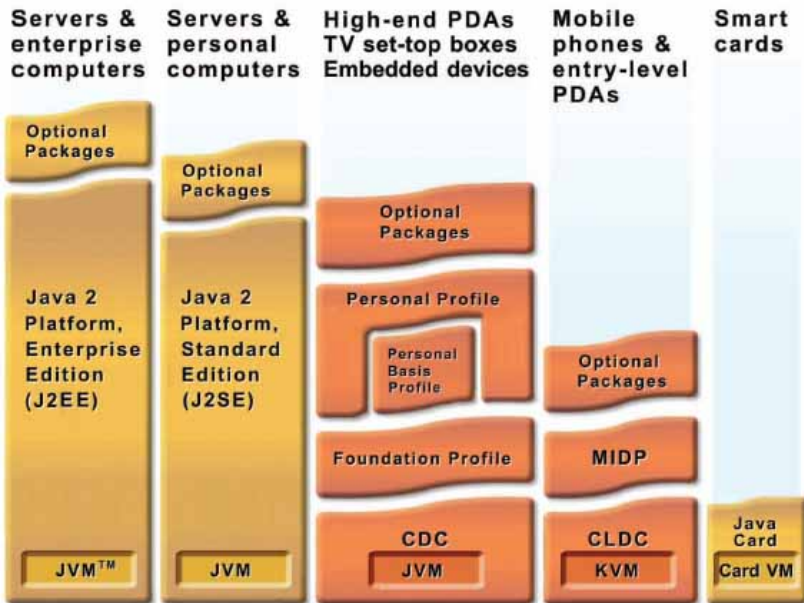


# Java 2 Micro Edition

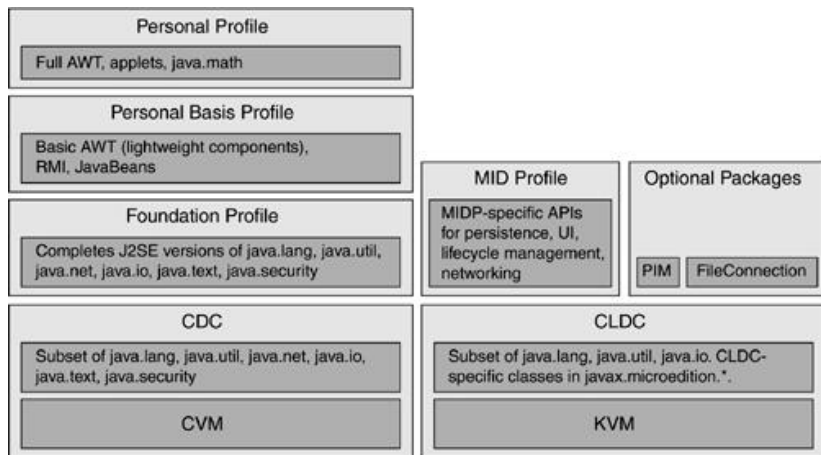
- A specification.
- A subset of Java Virtual Machine.
- A subset of standard Java library.
- **Many vendors.**



# Java 2 Micro Edition



# Java 2 Micro Edition



# Programming in J2ME

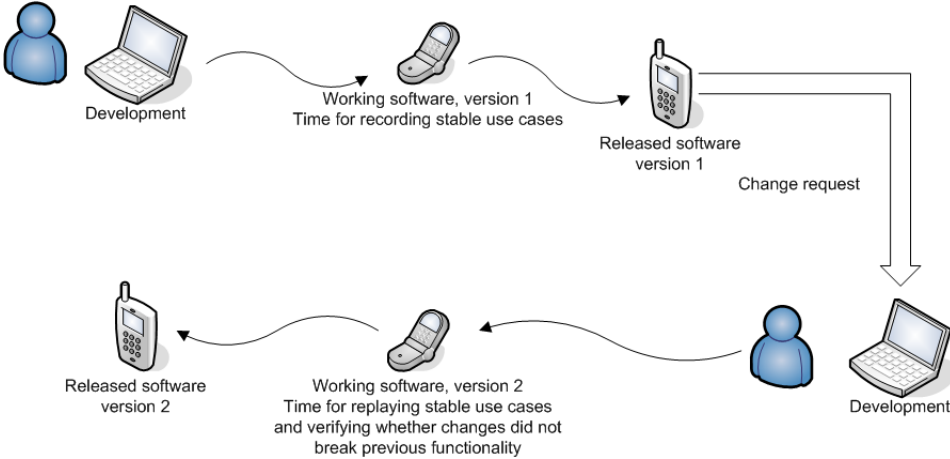
- Each mobile device has different hardware.
- Different KVM implementations (and bugs).
- “Standard” APIs implemented in different ways.
- A number of non-standard APIs and proprietary solutions.
- No system-level support for application testing.

**Conclusion:**  
**programming and testing is difficult in J2ME.**

# Project scope

- ① Focus on capture-replay tests (GUI and other events).
- ② Should facilitate integration and regression tests in J2ME.
- ③ Should work on emulators and actual devices.

# Example of use



Capture-replay and regression testing.

## Related projects

- J2ME Unit
- Sony-Ericsson Mobile JUnit
- Motorola Gatling
- CLDC Unit
- IBM Rational Test RT
- Research In Motion – BlackBerry Fledge emulator

And the ultimate answer is . . .

# RobotME

(of course the ultimate question still being “what’s  $6 \times 9$ ?”)

## The core idea

- Follow the regular capture-replay pattern.
- Cater for missing “robot” API by modifying the software at the **bytecode** level.
- Verify replay-phase correctness by analysis of captured events.



## Dynamic code injection

- Identify places which should generate an event (“injection points”).
- Intercept parameters at injection points, injecting custom proxies.

## Injection points: subclassing

Custom inheritance from system classes (subclassing).  
Form, Canvas, MIDlet...

```
1 public class MyMidlet extends MIDlet {  
2     protected void startApp() throws MIDletStateChangeException {  
3         // application code here.  
4     }  
5 }
```

We need to intercept the call to `startApp()` method.

- Make MyMidlet a subclass of RobotMIDlet?

```
1 public class MyMidlet extends RobotMIDlet {
```

all methods virtual, multi-level inheritance

- Make MyMidlet a subclass of RobotMIDlet?

```
1 public class MyMidlet extends RobotMIDlet {
```

all methods virtual, multi-level inheritance

- Make a custom subclass RobotMIDlet\$1 extend MyMidlet?

```
1 public class RobotMIDlet$1 extends MyMidlet {
```

finalized classes, multi-level inheritance

- Make MyMidlet a subclass of RobotMIDlet?

```
1 public class MyMidlet extends RobotMIDlet {
```

all methods virtual, multi-level inheritance

- Make a custom subclass RobotMIDlet\$1 extend MyMidlet?

```
1 public class RobotMIDlet$1 extends MyMidlet {
```

finalized classes, multi-level inheritance

- Use delegation pattern and alter the code of startApp()?

```
1 public class MyMidlet extends MIDlet {
2     protected void startApp() throws MIDletStateChangeException {
3         RobotME.event(this, STARTAPP_BEFORE);
4         try {
5             // application code here.
6         } finally {
7             RobotME.event(this, STARTAPP_AFTER);
8         }
9     }
10 }
```

code bloat, goto, architectural flaws

## Injection points: references

Direct use of an object (reference tracking).  
Command, Item

```
1 public class MyApplication {
2     private static final Command MY_COMMAND =
3         new Command("Cmd label", Command.OK, 1);
4
5     public MyApplication() {
6         Form f = new Form("Title");
7         f.addCommand(MY_COMMAND);
8     }
9 }
```

We need to track the identity (and value) of commands.

- Substitute all constructors of Form with a custom proxy?

```
1 Form f = new RobotME$Form("Title");
```

clashes with custom inheritance

- Substitute all constructors of Form with a custom proxy?

```
1 Form f = new RobotME$Form("Title");
```

clashes with custom inheritance

- Intercept all calls to addCommand()?

```
1 f.addCommand(RobotME.addCommand(f, MY_COMMAND));
```

not too bad



## Injection points: listeners

Listeners (system callbacks).

ItemStateListener, CommandListener, ItemCommandListener

```
1 public class MyForm extends Form implements CommandListener {
2     private final Command ADD_COMMAND;
3
4     public MyClass {
5         this.ADD_COMMAND = new Command("ADD", Command.OK, 1);
6         this.setCommandListener(this);
7     }
8
9     public void commandAction(Command c, Displayable d) {
10         // event handling code.
11         if (c == ADD_COMMAND) {
12             // ...do something.
13         }
14     }
15 ...
```

We need to track (and stimulate) commands for the listener. Note there is only **one** listener on a Form.

- Intercept all calls to `setCommandListener()`?

```
1 this.setCommandListener(RobotME.setCommandListener(this, command));
```

What if somebody uses "==" to compare listeners?

- Intercept all calls to `setCommandListener()`?

```
1 this.setCommandListener(RobotME.setCommandListener(this, command));
```

What if somebody uses "==" to compare listeners?

- How to remember the command received (if it's a dynamic reference)?

The reference changes between runs.

- Intercept all calls to `setCommandListener()`?

```
1 this.setCommandListener(RobotME.setCommandListener(this, command));
```

What if somebody uses "==" to compare listeners?

- How to remember the command received (if it's a dynamic reference)?

The reference changes between runs.

- How to generate an identical event dynamically?

The event should match the original Display/ Form pair.

## Examples of injected code

Original code (fragment):

```
1 final Form form = new Form("Questionnaire");
2 ...
3 form.addCommand(CMD1_EXIT);
4 form.addCommand(CMD2_OK);
5 form.setCommandListener(this);
6
7 Display.getDisplay(this).setCurrent(form);
```

## Code modified for the recording phase:

```
1 Form form = new Form("Questionnaire");
2 ...
3 form.addCommand(b); // NOTE: variable names have been obfuscated.
4 RobotMERRecorder.getRecorderInstance().commandAddedToDisplayable(b, form);
5 form.addCommand(c);
6 RobotMERRecorder.getRecorderInstance().commandAddedToDisplayable(c, form);
7 form.setCommandListener(this);
8 Display.getDisplay(this).setCurrent(form);
9 RobotMERRecorder.getRecorderInstance().setCurrentDisplayable(form);
```

## Code modified for the replay phase:

```
1 Form form = new Form("Questionnaire");
2 form.addCommand(b);
3 RobotMERReplaying.getReplayingInstance().commandAddedToDisplayable(b, form);
4 form.addCommand(c);
5 RobotMERReplaying.getReplayingInstance().commandAddedToDisplayable(c, form);
6 form.setCommandListener(this);
7 RobotMERReplaying.getReplayingInstance()
8     .commandListenerSetOnDisplayable(this, form);
9 Display.getDisplay(this).setCurrent(form);
10 RobotMERReplaying.getReplayingInstance().setCurrentDisplayable(form);
11
12 RobotMERReplaying.getReplayingInstance().startReplaying();
```

A teraz przerwa dla miłośników granul.

A teraz przerwa dla miłośników granul.





At the bytecode level  
Java can be quite pleasant (and surprising!)

# Java in assembler mode ;)

- The stack.

## Java in assembler mode ;)

- The stack.
- Local variables.

# Java in assembler mode ;)

- The stack.
- Local variables.
- Opcodes and their mnemonics.

# Java in assembler mode ;)

- The stack.
- Local variables.
- Opcodes and their mnemonics.
- Code verification.

# Reverse-engineering Java code

```
1 public static final void method(int i)
2 {
3     System.out.println(i);
4 }
```



# Reverse-engineering Java code

```
1 public static final void method(int i)
2 {
3     System.out.println(i);
4 }
```



```
1 public static final void method(int i)
2 {
3     getstatic #16 <Field PrintStream System.out>
4     iload_0
5     invokevirtual #22 <Method void PrintStream.println(int)>
6     return
7 }
```

## Reverse-engineering Java code

```
1  private final long sum(int a, int b) {  
2      return a + b;  
3  }  
4  
5  public final void method() {  
6      System.out.println(sum(2, 50));  
7  }
```



## Reverse-engineering Java code

```
1 private final long sum(int a, int b) {
2     return a + b;
3 }
4
5 public final void method() {
6     System.out.println(sum(2, 50));
7 }
```

```
1 private final long sum(int a, int b) {
2     iload_1
3     iload_2
4     iadd
5     i2l
6     lreturn
7 }
8 public final void method() {
9     getstatic #20 <Field PrintStream System.out>
10    aload_0
11    iconst_2
12    bipush 50
13    invokespecial #26 <Method long sum(int, int)>
14    invokevirtual #28 <Method void PrintStream.println(long)>
15    return
16 }
```

## Reverse-engineering Java code

```
1  public final void method(int i) {
2      switch (i) {
3          case 1:
4          case 25:
5          case -5:
6          case 1128:
7              break;
8          default:
9              throw new RuntimeException();
10     }
11 }
```

# Reverse-engineering Java code

```
1  public final void method(int i) {
2      switch (i) {
3          case 1:
4          case 25:
5          case -5:
6          case 1128:
7              break;
8          default:
9              throw new RuntimeException();
10     }
11 }
```

```
1  public final void method(int i)
2  {
3      0 0:iload_1
4      1 1:lookupswitch default 47
5          -5: 44
6          1: 44
7          25: 44
8          1128: 44
9      2 44:goto 55
10     3 47:new #16 <Class RuntimeException>
11     4 50:dup
12     5 51:invokespecial #18 <Method void RuntimeException()>
13     6 54:athrow
```

## Reverse-engineering Java code

```
1  public final int method(int i) {  
2      try {  
3          if (i == 0) {  
4              return 0;  
5          }  
6          if (i == 1) {  
7              return 1;  
8          }  
9          if (i == 2) {  
10             return 2;  
11         }  
12     } finally {  
13         System.out.println("aaa");  
14         System.out.println("bbb");  
15         System.out.println("ccc");  
16     }  
17     return -1;  
18 }
```

What will this compile into?

```
1  public final int method(int i) {
2      if (i == 0) {
3          System.out.println("aaa"); System.out.println("bbb");
4          System.out.println("ccc");
5          return 0;
6      }
7      if (i == 1) {
8          System.out.println("aaa"); System.out.println("bbb");
9          System.out.println("ccc");
10         return 1;
11     }
12     if (i == 2) {
13         System.out.println("aaa"); System.out.println("bbb");
14         System.out.println("ccc");
15         return 2;
16     } else {
17         System.out.println("aaa"); System.out.println("bbb");
18         System.out.println("ccc");
19         return -1;
20     }
21
22 exception_handler:
23     System.out.println("aaa"); System.out.println("bbb");
24     System.out.println("ccc");
25     throw exception;
26 }
```

```
1  public final int method(int i) {
2      int j;
3      if (i != 0)
4          goto _L13;
5      jsr local;
6      j = 0;
7      return j;
8  _L13:
9      if (i != 1) goto _L27;
10     jsr local;
11     j = 1;
12     return j;
13  _L27
14     if (i != 2) goto _L2;
15     jsr local;
16     j = 2;
17     return j;
18  _L2:
19     jsr local;
20     return -1;
21  local:
22     System.out.println("aaa");
23     System.out.println("bbb");
24     System.out.println("ccc");
25     ret;
26 }
```

```
    form.setCommandListener(this);
// 37 78:aload_1
// 38 79:aload_0
// 39 80:invokevirtual #49 <Method void Displayable.setCommandListener(CommandListener)>
    RobotMEReplaying.getReplayingInstance().commandListenerSetOnDisplayable(this, form);
// 40 83:invokestatic #45 <Method RobotMEReplaying RobotMEReplaying.getReplayingInstance()>
// 41 86:aload_0
// 42 87:aload_1
// 43 88:invokevirtual #40 <Method void RobotMEReplaying.commandListenerSetOnDisplayable(Comm
    Display.getDisplay(this).setCurrent(form);
// 44 91:aload_0
// 45 92:invokestatic #43 <Method Display Display.getDisplay(MIDlet)>
// 46 95:aload_1
// 47 96:invokevirtual #50 <Method void Display.setCurrent(Displayable)>
    RobotMEReplaying.getReplayingInstance().setCurrentDisplayable(form);
// 48 99:invokestatic #45 <Method RobotMEReplaying RobotMEReplaying.getReplayingInstance()>
// 49 102:aload_1
// 50 103:invokevirtual #51 <Method void RobotMEReplaying.setCurrentDisplayable(Displayable)>
}
RobotMEReplaying.getReplayingInstance().startReplaying();
// 51 106:invokestatic #45 <Method RobotMEReplaying RobotMEReplaying.getReplayingInstance()>
// 52 109:invokevirtual #58 <Method void RobotMEReplaying.startReplaying()>
// 53 112:return
```

```
/**
 * Visit a method and check if we need to create a delegation stub.
 */
public MethodVisitor visitMethod(int access, String name, String desc, String signature, String[] exceptions) {
    final MethodVisitor mw = super.visitMethod(access, name, desc, signature, exceptions);
    if (METHOD_NAME_CONSTRUCTOR.equals(name) && MIDDLELET_CLASS_NAME.equals(superClassName)) {
        setProcessing(true);
        return new MethodAdapter(mw) {
            @Override
            public void visitInsn(int opcode) {
                // if last statement in constructor:
                if (Opcodes.RETURN == opcode) {
                    // original source code:
                    // RobotMERecorder.getInstance().setMIDlet(this);
                    final String internalRobotMeClassName = Type.getInternalName(getInternalRobotMeClassName());
                    final String methodDescriptor = Type.getMethodDescriptor(Type
                        .getType(getInternalRobotMeClassName()), new Type[0]);
                    mv.visitMethodInsn(Opcodes.INVOKESTATIC, internalRobotMeClassName, getFactoryMethodName(),
                        methodDescriptor);
                    mv.visitVarInsn(Opcodes.ALOAD, 0);
                    mv.visitMethodInsn(Opcodes.INVOKEVIRTUAL, internalRobotMeClassName, "setMIDlet",
                        "(Ljavax/microedition/midlet/MIDlet;)V");
                }
                super.visitInsn(opcode);
            }
        };
    } else {
        return mw;
    }
}
```

ASMLib is used for preprocessing bytecode (statically).



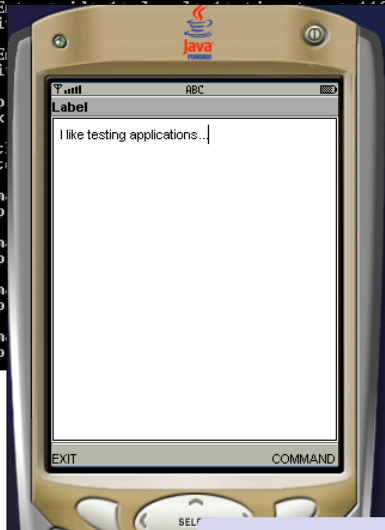
# Test maintenance

Maintenance through **human-comprehensible test scripts**.

```
1 <scenario>
2   <event timestamp="1000">
3     <displayable-changed title="Hello screen" type="TEXTBOX" />
4   </event>
5
6   <event timestamp="2000">
7     <command cmdLabel="Start app" displayableTitle="Hello screen" />
8   </event>
9
10  <event timestamp="3000">
11    <textbox-modification assertion="true" strongAssertion="true"
12      string="I like testing" />
13  </event>
14 </scenario>
```

# Time for a live demo!

```
[javal Received logEntryId: 1
[javal class: org.robotme.core.log.entries.LogEntry; id: 1; level: 1; timestamp: 11660
e; msg: MIDlet set to: org.example.midlet.TestTextBoxMIDlet@d590dbc; ex:
[javal Received logEntryId: 1
[javal class: org.robotme.core.log.entries.LogEntry; id: 1; level: 1; timestamp: 11660
e; msg: Command added to displayable: javax.microedition.lcdui.Displayable; ex:
[javal Received logEntryId: 1
[javal class: org.robotme.core.log.entries.LogEntry; id: 1; level: 1; timestamp: 11660
e; msg: Command added to displayable: javax.microedition.lcdui.Displayable; ex:
[javal Received logEntryId: 3
[javal class: org.robotme.core.log.entries.Displayable; id: 3; level: 1; timestamp: 11660
lse; assertion: true; msg: Displayable set to: javax.microedition.lcdui.Displayable; ex:
[javal Received logEntryId: 4
[javal class: org.robotme.core.log.entries.TextEntry; id: 4; level: 1; timestamp: 11660
true; assertion: false; msg: ; ex: ; string: I like testing applications...;
[javal Received logEntryId: 2
[javal class: org.robotme.core.log.entries.Command; id: 2; level: 1; timestamp: 11660
on: false; msg: Command invoked: COMMAND; ex: ; displayable: javax.microedition.lcdui.Displayable;
[javal Received logEntryId: 2
[javal class: org.robotme.core.log.entries.Command; id: 2; level: 1; timestamp: 11660
on: false; msg: Command invoked: COMMAND; ex: ; displayable: javax.microedition.lcdui.Displayable;
[javal Received logEntryId: 2
[javal class: org.robotme.core.log.entries.Command; id: 2; level: 1; timestamp: 11660
on: false; msg: Command invoked: COMMAND; ex: ; displayable: javax.microedition.lcdui.Displayable;
[javal Received logEntryId: 2
[javal class: org.robotme.core.log.entries.Command; id: 2; level: 1; timestamp: 11660
on: false; msg: Command invoked: COMMAND; ex: ; displayable: javax.microedition.lcdui.Displayable;
[javal Received logEntryId: 2
```



Server console.

Emulator window.

## Summary

- Testing is difficult in J2ME.
- Bytecode manipulation can provide a substitute for the required API functions.
- The prototype a bit gritty, but functional.

## Little victories

- Springer LNCS publication (10th BIS conference).
- UAM Foundation — “Pomysł na biznes” competition.

Thank you for your attention.