DEBUGGING EN LOGGING

W. Schepens

nov. 2011

println

`System.out.println(…)` `System.err.println(…)`

- Normale output van console-programma
- Info over programmaverloop
- Om te fouten op te sporen (debugging)
  - Geraak ik hier?
  - Waarde van variabelen

- Waar komt dit terecht bij GUI? Bij Web-app?
- Enkel tijdens development of ook in production code?
- Achteraf verwijderen? In commentaar?

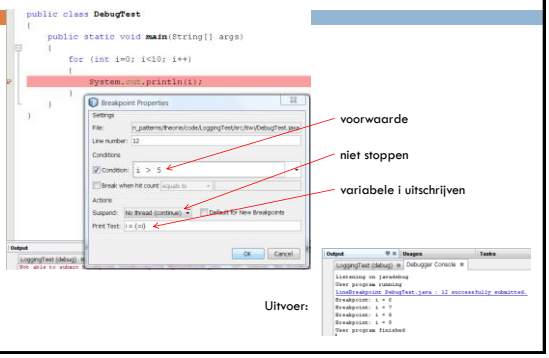
DEBUGGING

Debugging

- `println(…)`: **intrusief**!
- Vermijd wijzigingen in code met als enig doel debuggen (zeker third-party code)

- Beter:
  - breakpoints
  - step over, step into, step out, continue, …
  - inspect variables, watch expression

Breakpoints in NetBeans



Soorten breakpoints in NetBeans

- **Line.**
- **Class.** You can break when the class is loaded into the virtual machine, unloaded from the virtual machine, or both.
- **Exception.** You can break whenever a specific exception is caught, whenever a specific exception is not handled in the source code, or whenever any exception is encountered regardless of whether the program handles the error or not.
- **Field.** You can stop execution of your program whenever a field in a specific class is accessed (for example, the method was called with the variable as an argument), modified or both.
- **Method.** Program execution stops every time the method is entered, exited or both.
- **Thread.** You can break program execution whenever a thread starts, stops, or both.

Zie NetBeans Help

Ctrl-Shift-F8: New breakpoint Alt-Shift-5: Show breakpoints

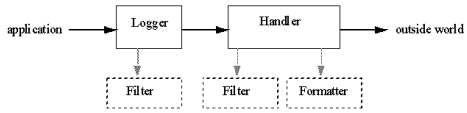
LOGGING

Logging

Gebruik beter een **logging-library** zoals
- `java.util.logging`
- log4j

Voordelen:
- Levels bv. info, warning, …
- Filters
- Formatters
- Handlers/output-channels bv. naar bestand, DB, e-mail, …
- Configuratie door client bv. via XML-bestand

java.util.logging



java.util.logging: logger-object

```
Logger logger = Logger.getLogger(name)
```

Hierbij is `name` dot-separated, bv. `be.tiwi.MyApp`

Typisch:

```
name = ThisClass.class.getName()
name = this.getClass().getName()
```

- Gecached
- Bij creatie: configuratie via `LogManager`

java.util.logging: levels

Standaard gebruik:

```
logger.log(Level level, String message)
```

Met parameters:

```
logger.log(level, msg, param), …
```

Levels:
- FINE (500)
- CONFIG (700)
- INFO(800)
- WARNING(900)
- SEVERE (1000)

Convenience:

```
logger.info(msg),  logger.warning(msg), …
```

java.util.logging: handlers

ConsoleHandler
writes to System.err

StreamHandler
writes to an OutputStream.

FileHandler
writes to a single file, or to a set of rotating log files.

SocketHandler
writes to remote TCP ports.

MemoryHandler
buffers log records in memory.

java.util.logging: formatters

SimpleFormatter

```
24-nov-2011 9:48:29 LogTest main
WARNING: This is a warning
```

XMLFormatter

```
<record>
  <date>2011-11-24T09:52:18</date>
  <millis>1322124738566</millis>
  <sequence>1</sequence>
  <logger>LogTest</logger>
  <level>WARNING</level>
  <class>LogTest</class>
  <method>main</method>
  <thread>10</thread>
  <message>This is a warning</message>
</record>
```

java.util.logging: i18n

Internationalisation/localisation (i18n, l12n)

-> resourcebundle meegeven aan logger

java.util.logging: configuratie

Hiërarchische namespace: elke logger

- erft eigenschappen van zijn ouder
- kan zelf overriden

Configuratie in code, bv.

```
FileHandler fileHandler = new fileHandler("%t/mylog.txt");
fileHandler.setFormatter(new SimpleFormatter());
logger.addHandler(fileHandler);
```

Configuratie uit bestand

```
java -Djava.util.logging.config.file=logfile app
```

java.util.logging: configuratie

src/logging.properties:

```
handlers = java.util.logging.ConsoleHandler,
           java.util.logging.FileHandler

.level = ALL

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter =
   java.util.logging.SimpleFormatter

java.util.logging.FileHandler.level = ALL
java.util.logging.FileHandler.pattern = %t/LoggingTest.log

tiwi.level = WARNING
```

Andere logging-libraries

Alternatieven voor java.util.logging (JUL):

- Apache Commons Logging
- Simple Logging Facade for Java (SLF4J)
- Log4j

Waarom?

- Meer handlers (bv. E-mail, …)
- Betere formatting
- Beter configureerbaar

http://en.wikipedia.org/wiki/Java_Logging_Frameworks