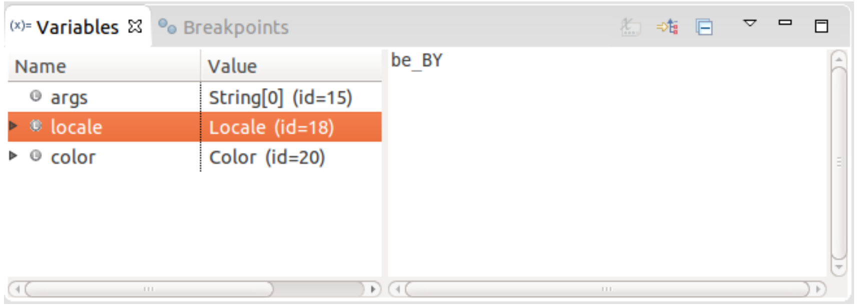
Templated visualization of object state with Vebugger

Daniel Rozenberg
Ivan Beschastnikh

Computer Science
University of British Columbia

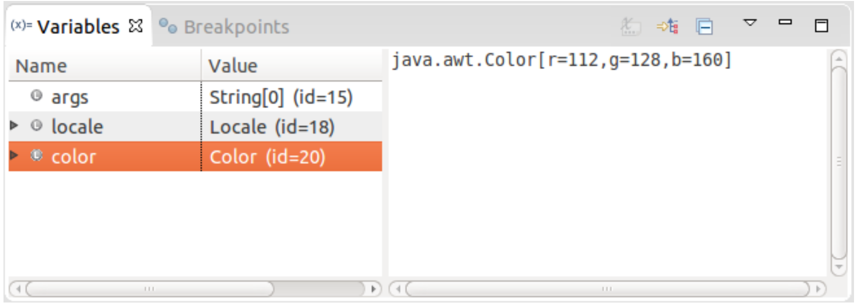
State inspection

Mind the abstraction gap



Name	Value
args	String[0] (id=15)
locale	Locale (id=18)
color	Color (id=20)

```
be_BY
```

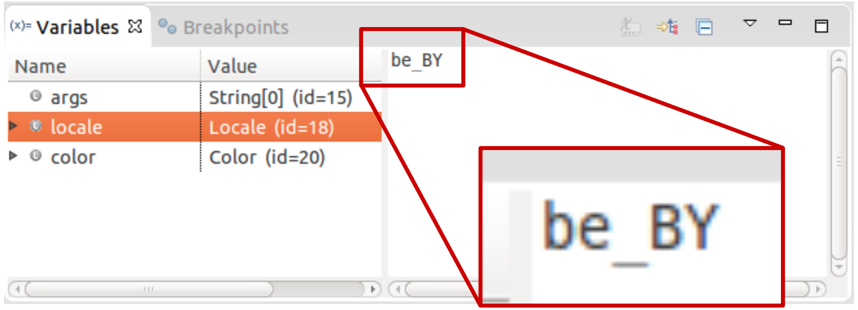


Name	Value
args	String[0] (id=15)
locale	Locale (id=18)
color	Color (id=20)

```
java.awt.Color[r=112,g=128,b=160]
```

State inspection — still hard!

Mind the abstraction gap



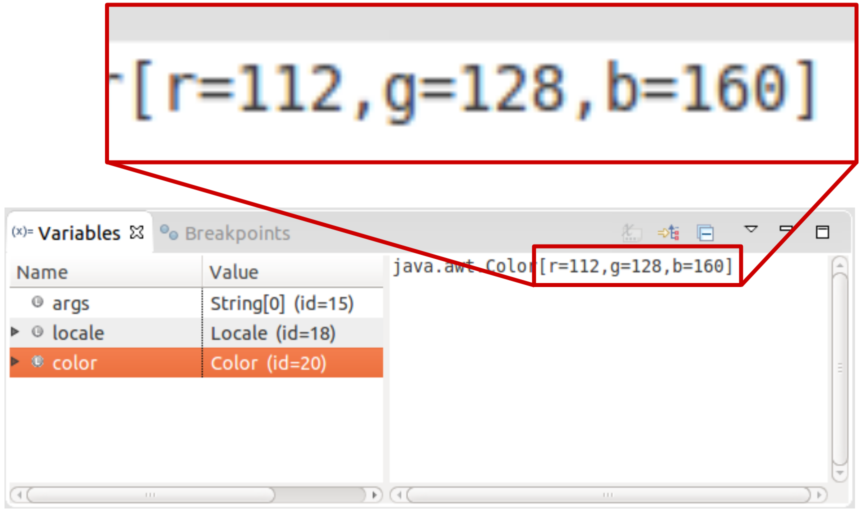
Variables | Breakpoints

Name	Value
args	String[0] (id=15)
locale	Locale (id=18)
color	Color (id=20)

be_BY

be_BY

Which **language** and **country** are represented here?



[r=112,g=128,b=160]

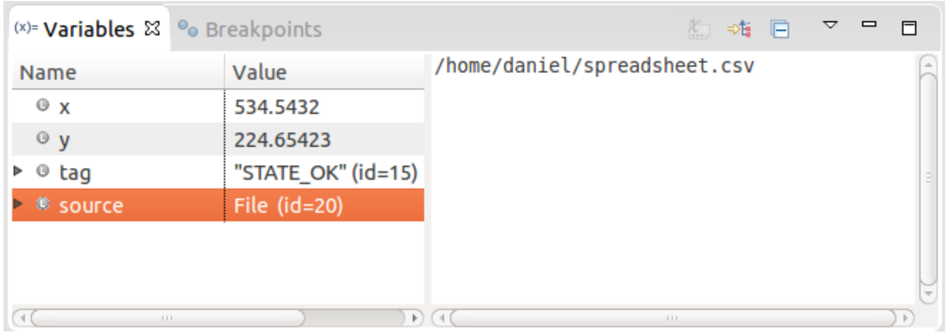
Variables | Breakpoints

Name	Value
args	String[0] (id=15)
locale	Locale (id=18)
color	Color (id=20)

java.awt.Color[r=112,g=128,b=160]

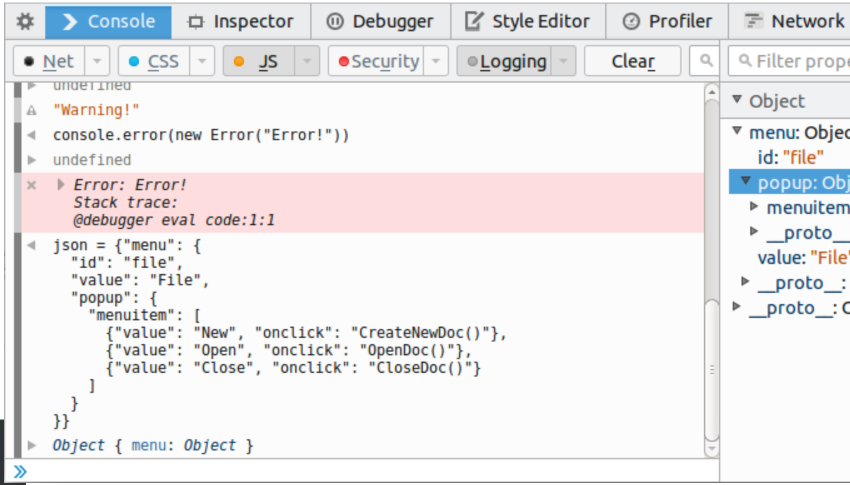
What **color** is this?

What developers use today



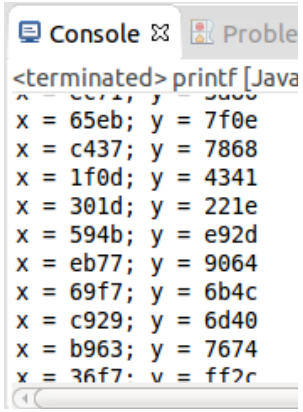
Name	Value
x	534.5432
y	224.65423
tag	"STATE_OK" (id=15)
source	File (id=20)

```
/home/daniel/spreadsheet.csv
```

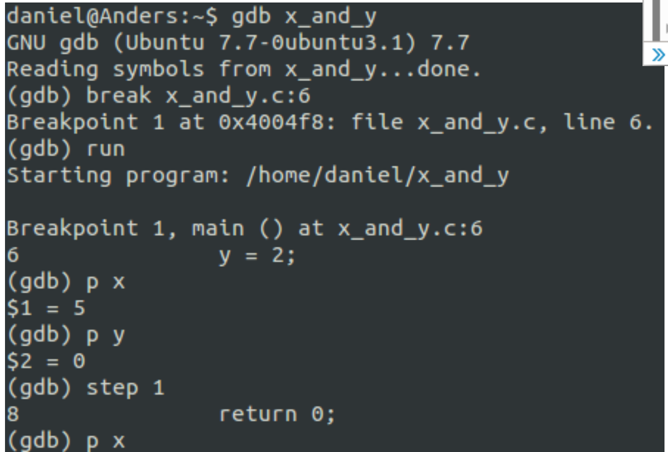


```
undefined
"Warning!"
console.error(new Error("Error!"))
undefined
Error: Error!
Stack trace:
@debugger eval code:1:1
json = {"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
Object { menu: Object }
```

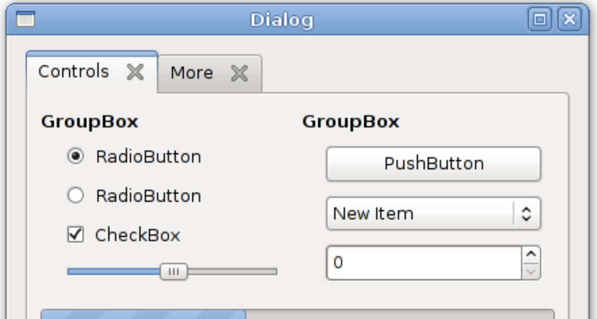
```
▼ Object
  ▼ menu: Objec
     id: "file"
    ▼ popup: Obj
      ▶ menuitem
      ▶ __proto__
     value: "File"
    ▶ __proto__:
  ▶ __proto__: O
```



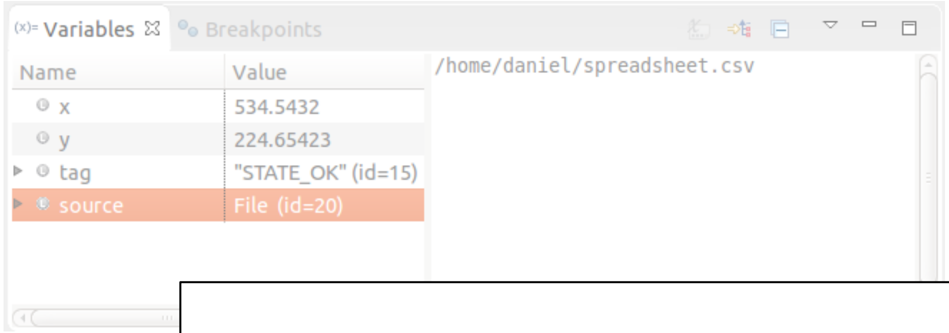
```
<terminated> printf [Java
x = 65eb; y = 7f0e
x = c437; y = 7868
x = 1f0d; y = 4341
x = 301d; y = 221e
x = 594b; y = e92d
x = eb77; y = 9064
x = 69f7; y = 6b4c
x = c929; y = 6d40
x = b963; y = 7674
x = 36f7; y = ff2c
```



```
daniel@Anders:~$ gdb x_and_y
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Reading symbols from x_and_y...done.
(gdb) break x_and_y.c:6
Breakpoint 1 at 0x4004f8: file x_and_y.c, line 6.
(gdb) run
Starting program: /home/daniel/x_and_y

Breakpoint 1, main () at x_and_y.c:6
6               y = 2;
(gdb) p x
$1 = 5
(gdb) p y
$2 = 0
(gdb) step 1
8               return 0;
(gdb) p x
```

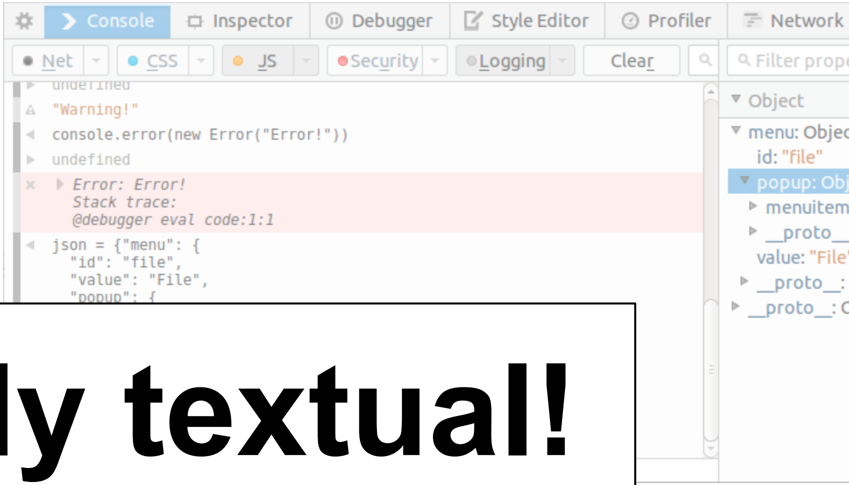


What developers use today



Name	Value
x	534.5432
y	224.65423
tag	"STATE_OK" (id=15)
source	File (id=20)

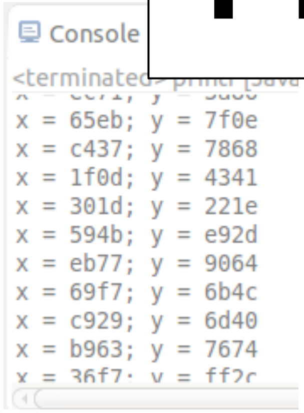
```
/home/daniel/spreadsheet.csv
```



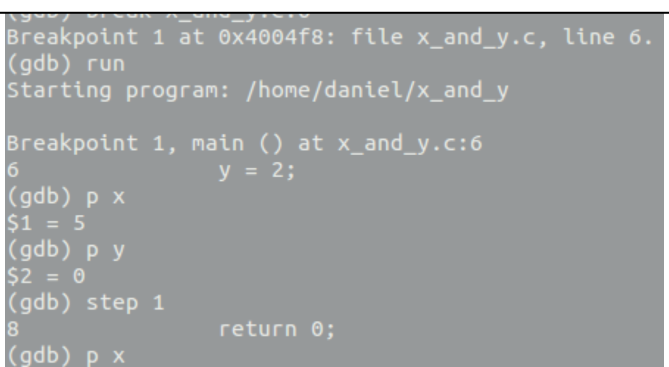
```
undefined
"Warning!"
console.error(new Error("Error!"))
undefined
Error: Error!
Stack trace:
@debugger eval code:1:1
json = {"menu": {
  "id": "file",
  "value": "File",
  "popup": {
```

```
Object
menu: Objec
  id: "file"
  popup: Obj
    menuitem
    __proto__
  value: "File"
  __proto__:
__proto__: O
```

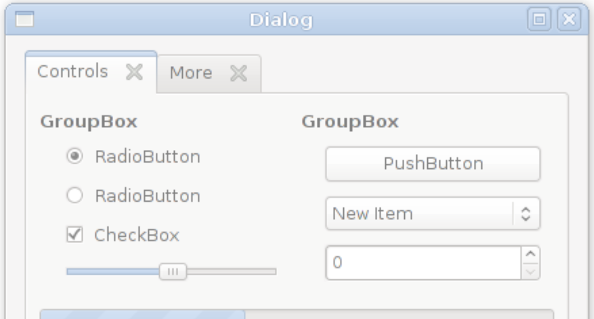
Predominantly textual!



```
<terminated> printf [Java
x = ce71; y = 5a00
x = 65eb; y = 7f0e
x = c437; y = 7868
x = 1f0d; y = 4341
x = 301d; y = 221e
x = 594b; y = e92d
x = eb77; y = 9064
x = 69f7; y = 6b4c
x = c929; y = 6d40
x = b963; y = 7674
x = 36f7; y = ff2c
```



```
(gdb) break x_and_y.c:6
Breakpoint 1 at 0x4004f8: file x_and_y.c, line 6.
(gdb) run
Starting program: /home/daniel/x_and_y

Breakpoint 1, main () at x_and_y.c:6
6               y = 2;
(gdb) p x
$1 = 5
(gdb) p y
$2 = 0
(gdb) step 1
8               return 0;
(gdb) p x
```



Our goal

Mitigate abstraction gap in state inspection

$$\begin{tikzcd}
\text{Textual} \arrow[d, "\textbf{Vebugger}"] \\
\text{Textual + Visual}
\end{tikzcd}$$

Design criteria

- Typed visualizations
  - Visualizations should be distinguished by classes
- Extensibility through templates
  - Easy to create templates
- IDE integration
  - Developers expect tools to be integrated into IDE
- **Do no harm**
  - Revert to default behavior on any failure

Design criteria — Vebugger

- Typed visualizations
  - Uses Java types to determine which template to use
- Extensibility through templates
  - Uses HTML+CSS
- IDE integration
  - Integrates into Eclipse’s “variable view” panel
- **Do no harm**
  - Displays the .toString() value when template missing

Demo time!

Future work

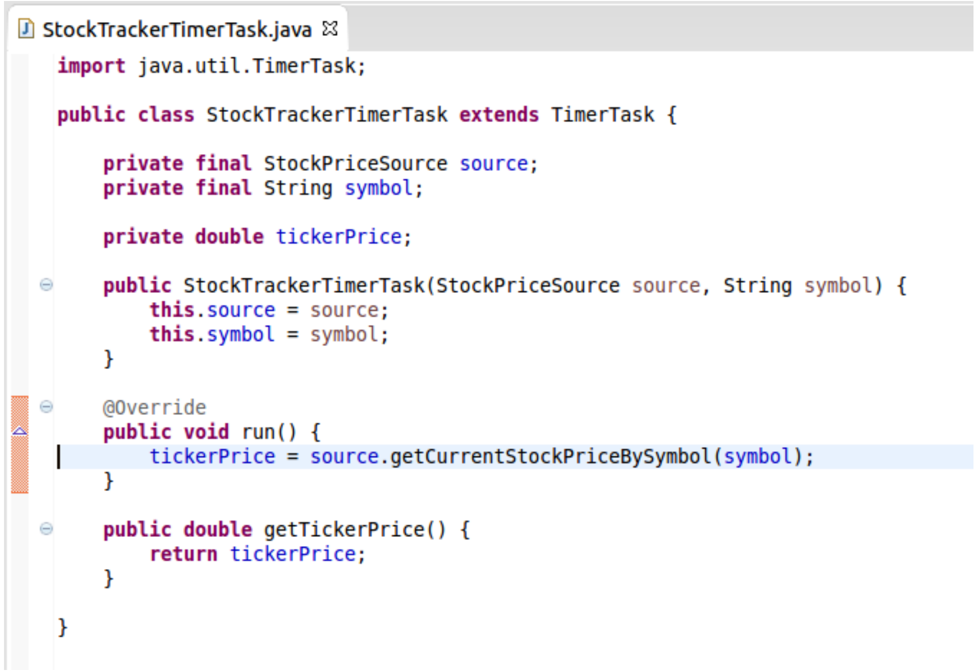
- Context-specific templates
- Navigation through visualizations
- Scalable visualizations
- Usability/viability user study
- Automating the template creation process
- Animating state transitions

Context-specific templates

Context could refer to:

- **Program domain**
- **Runtime environment**
- Developer’s task
- Operating system state
- etc…

Context *(domain)* templates

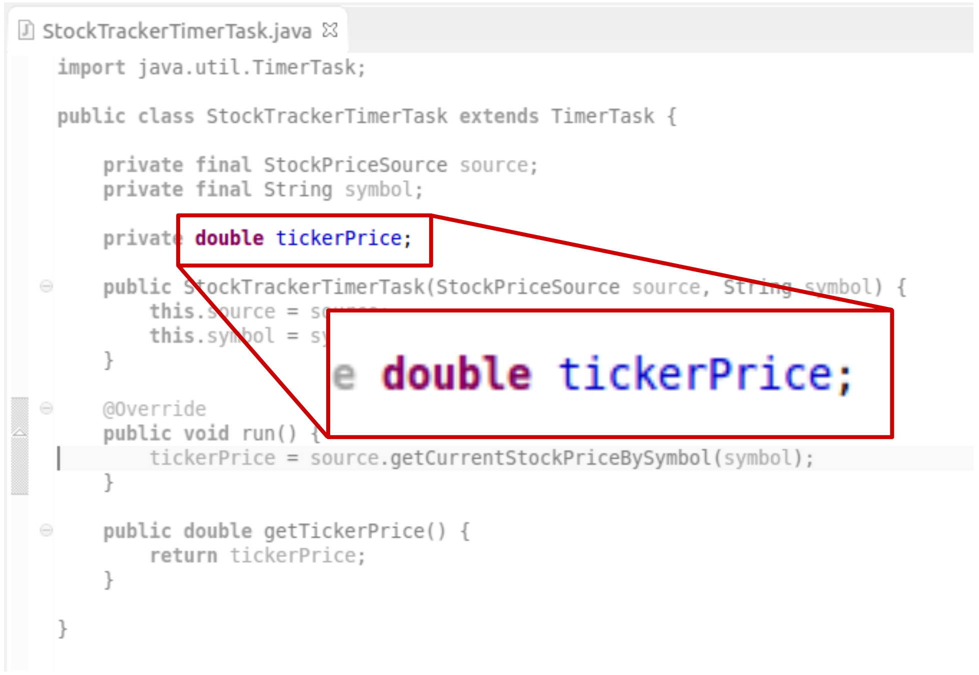


StockTrackerTimerTask.java

```java
import java.util.TimerTask;

public class StockTrackerTimerTask extends TimerTask {

    private final StockPriceSource source;
    private final String symbol;

    private double tickerPrice;

    public StockTrackerTimerTask(StockPriceSource source, String symbol) {
        this.source = source;
        this.symbol = symbol;
    }

    @Override
    public void run() {
        tickerPrice = source.getCurrentStockPriceBySymbol(symbol);
    }

    public double getTickerPrice() {
        return tickerPrice;
    }

}
```

Stock market application

Context *(domain)* templates



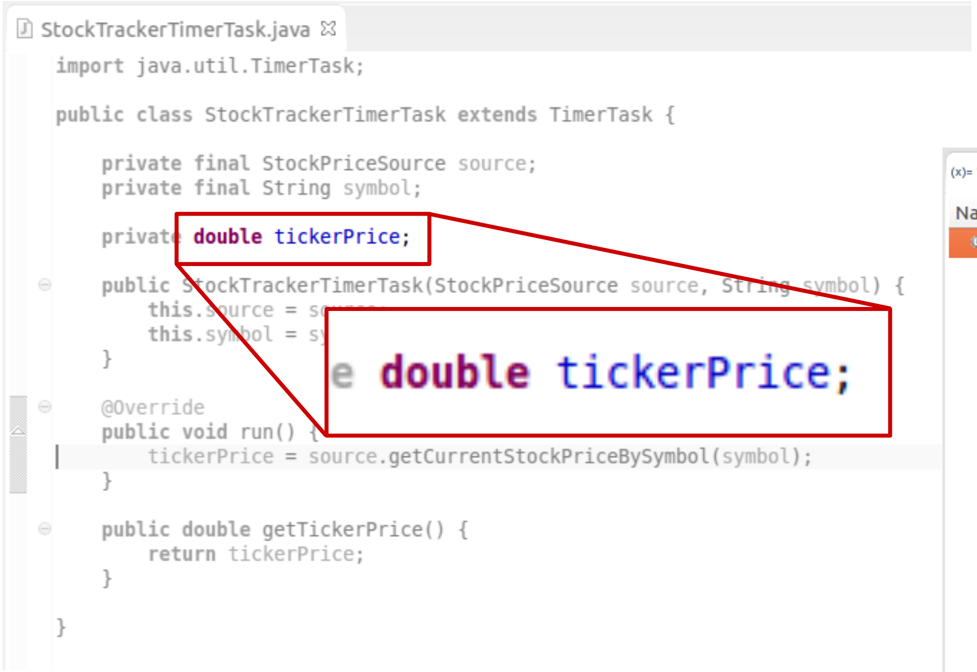
StockTrackerTimerTask.java

```
import java.util.TimerTask;

public class StockTrackerTimerTask extends TimerTask {

    private final StockPriceSource source;
    private final String symbol;

    private double tickerPrice;

    public StockTrackerTimerTask(StockPriceSource source, String symbol) {
        this.source = source;
        this.symbol = symbol;
    }

    @Override
    public void run() {
        tickerPrice = source.getCurrentStockPriceBySymbol(symbol);
    }

    public double getTickerPrice() {
        return tickerPrice;
    }

}
```

```
e double tickerPrice;
```

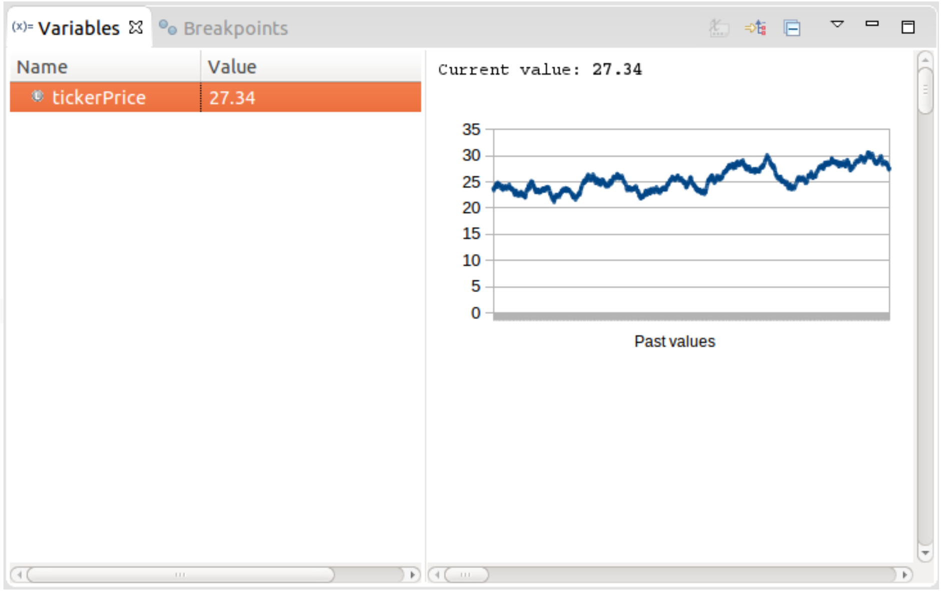
Stock market application

Context *(domain)* templates



```
StockTrackerTimerTask.java
import java.util.TimerTask;

public class StockTrackerTimerTask extends TimerTask {

    private final StockPriceSource source;
    private final String symbol;

    private double tickerPrice;

    public StockTrackerTimerTask(StockPriceSource source, String symbol) {
        this.source = source;
        this.symbol = symbol;
    }

    @Override
    public void run() {
        tickerPrice = source.getCurrentStockPriceBySymbol(symbol);
    }

    public double getTickerPrice() {
        return tickerPrice;
    }

}
```

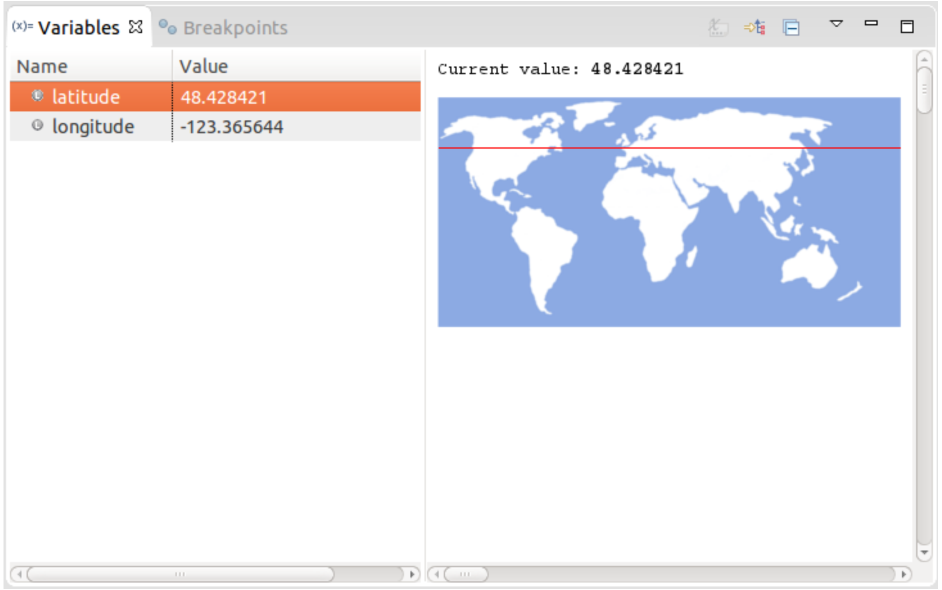
Stock market application



Name	Value
tickerPrice	27.34

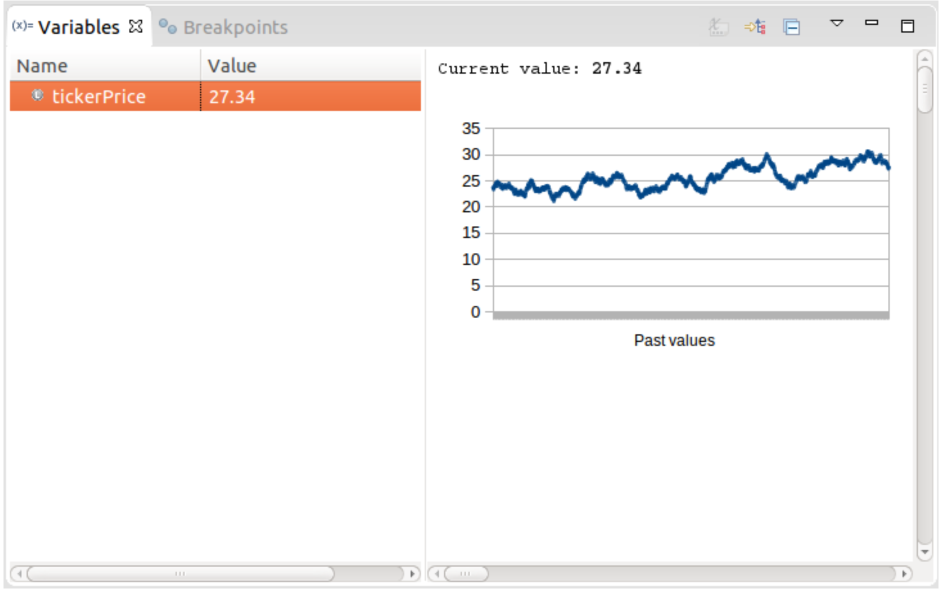
```
Current value: 27.34
```

Context *(domain)* templates



Name	Value
latitude	48.428421
longitude	-123.365644

```
Current value: 48.428421
```



Name	Value
tickerPrice	27.34

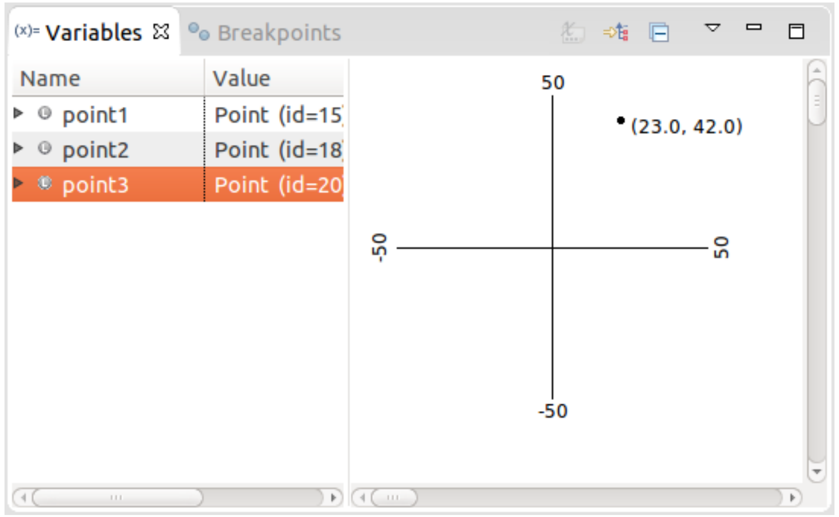
```
Current value: 27.34
```

Context *(domain)* templates

How to select a domain template?

- Manual user selection *simplest solution*
- Infer from variable names *static analysis*
- Dynamic object inspection *dynamic analysis*

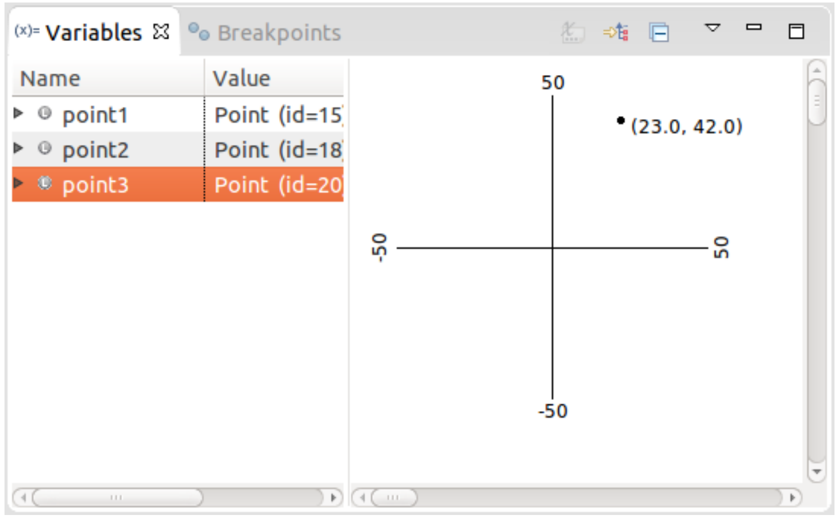
Context *(runtime)* templates



Name	Value
point1	Point (id=15
point2	Point (id=18
point3	Point (id=20

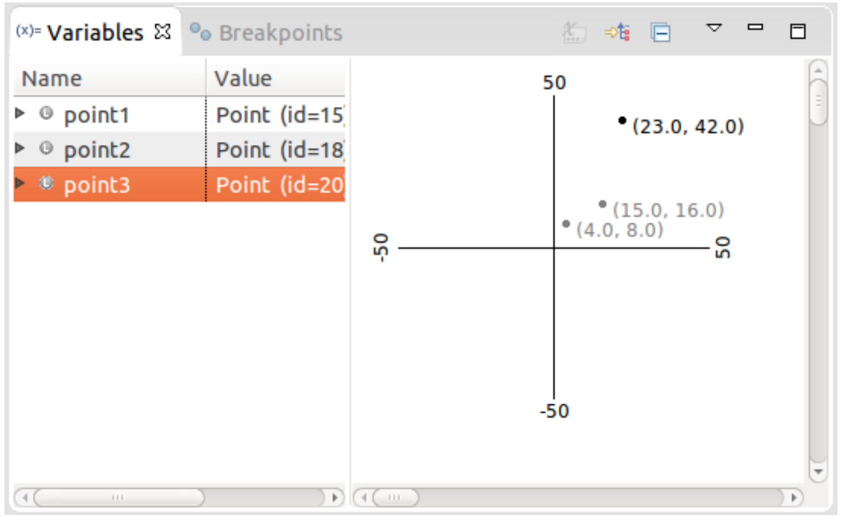
Without context

Context *(runtime)* templates



Name	Value
point1	Point (id=15
point2	Point (id=18
point3	Point (id=20

Without context



Name	Value
point1	Point (id=15
point2	Point (id=18
point3	Point (id=20

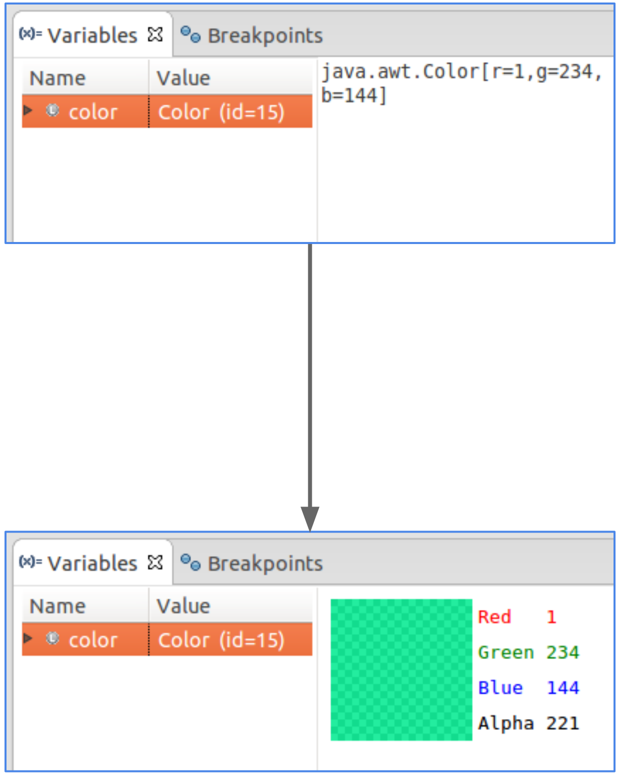
With context

Conclusion

Motivation: abstraction gap in state inspection

- Articulated design criteria for tools that expose object state for debugging purposes
- Built Vebugger, a framework for visualizing type-specific object state in Eclipse

Vebugger is free software!
https://github.com/daniboy/vebugger



Name	Value
color	Color (id=15)

```
java.awt.Color[r=1,g=234,
b=144]
```

Name	Value
color	Color (id=15)

```
Red     1
Green 234
Blue  144
Alpha 221
```

Backup slides

Related works

Other tools that show visual state

- T. D. Hendrix, J. H. Cross II, and L. A. Barowski. An extensible framework for providing dynamic data structure visualizations in a lightweight IDE. *ACM SIGCSE Bulletin*, 36(1):387–391, 2004
- C. Demetrescu and I. Finocchi. A data-driven graphical toolkit for software visualization. In *VISSOFT*, 2006
- B. Alsallakh, P. Bodesinsky, S. Miksch, and D. Nasseri. Visualizing Arrays in the Eclipse Java IDE. In *CSMR*, 2012

Exposing context-sensitive state

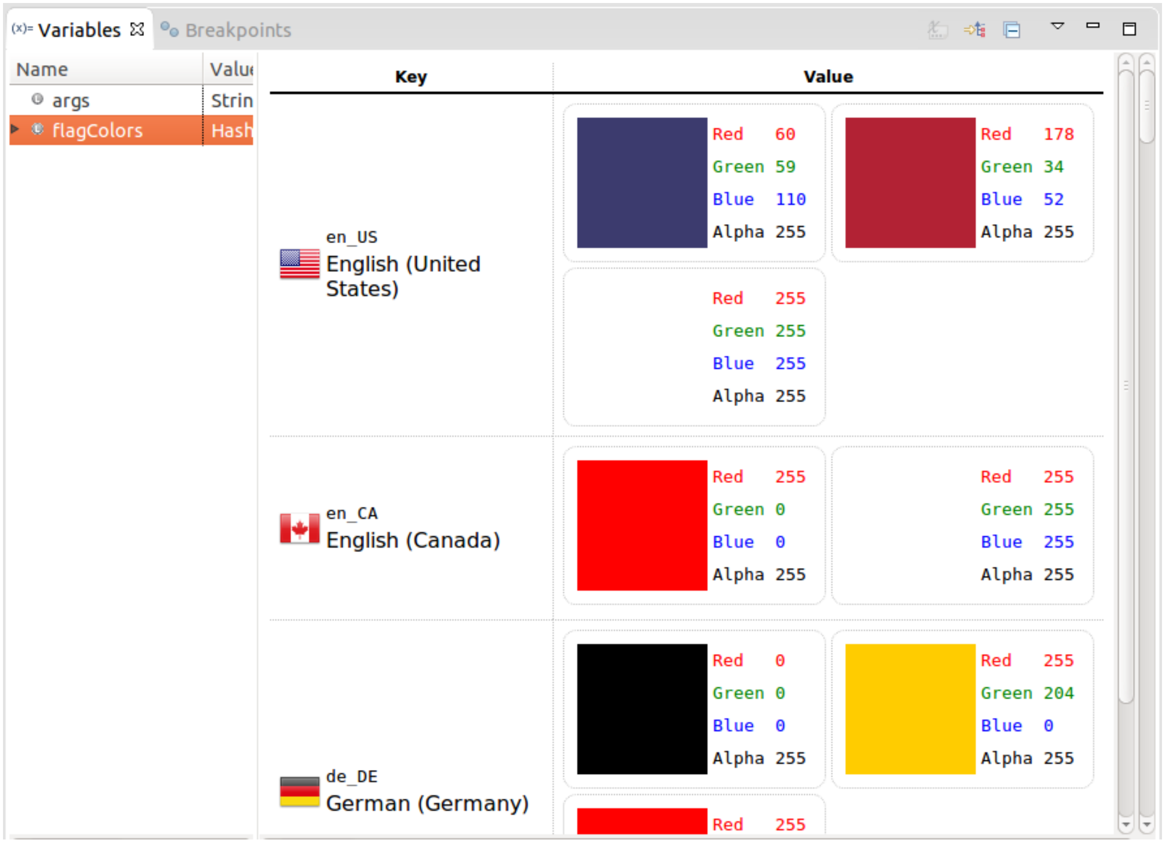
- D. A. Mellis. Tangible code. Master's thesis, Interaction Design Institute Ivrea, 2006
- F. Beck, F. Hollerich, S. Diehl, and D. Weiskopf. Visual monitoring of numeric variables embedded in source code. In *VISSOFT*, 2013
- A. J. Ko and B. A. Myers. Debugging reinvented. In *ICSE*, 2008

Limitations

- Heterogeneity — too many classes, too many contexts! How to streamline the template creation process to become a part of the debugging process?
- Scalability — exposing big-data without overwhelming the user or missing out on details. An open problem in Information Visualization.

Navigation with the aid of visualizations

Map<Locale, Set<Color>> flagColors = …

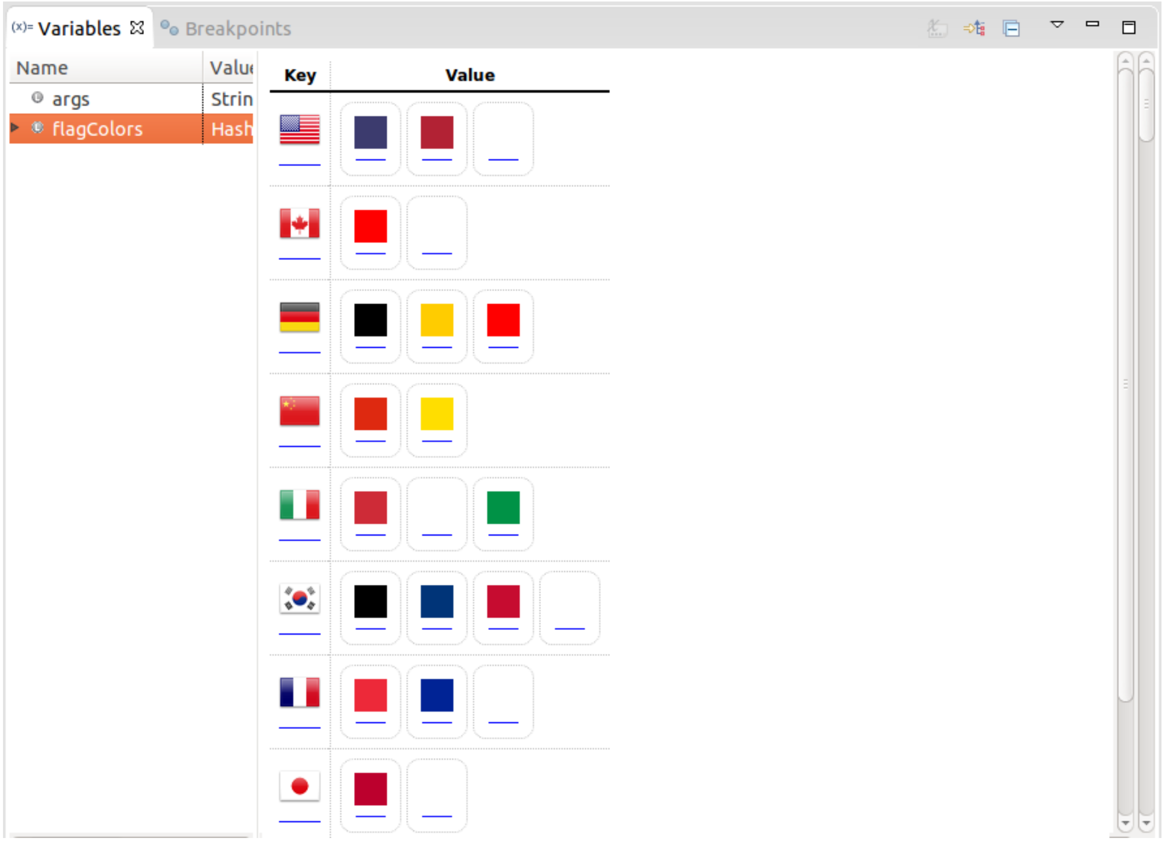


Name	Valu
args	Strin
flagColors	Hash

Key	Value	
en_US English (United States)	🟦 Red 60 Green 59 Blue 110 Alpha 255	🟥 Red 178 Green 34 Blue 52 Alpha 255
	⬜ Red 255 Green 255 Blue 255 Alpha 255	
en_CA English (Canada)	🟥 Red 255 Green 0 Blue 0 Alpha 255	⬜ Red 255 Green 255 Blue 255 Alpha 255
de_DE German (Germany)	⬛ Red 0 Green 0 Blue 0 Alpha 255	🟨 Red 255 Green 204 Blue 0 Alpha 255
	🟥 Red 255	

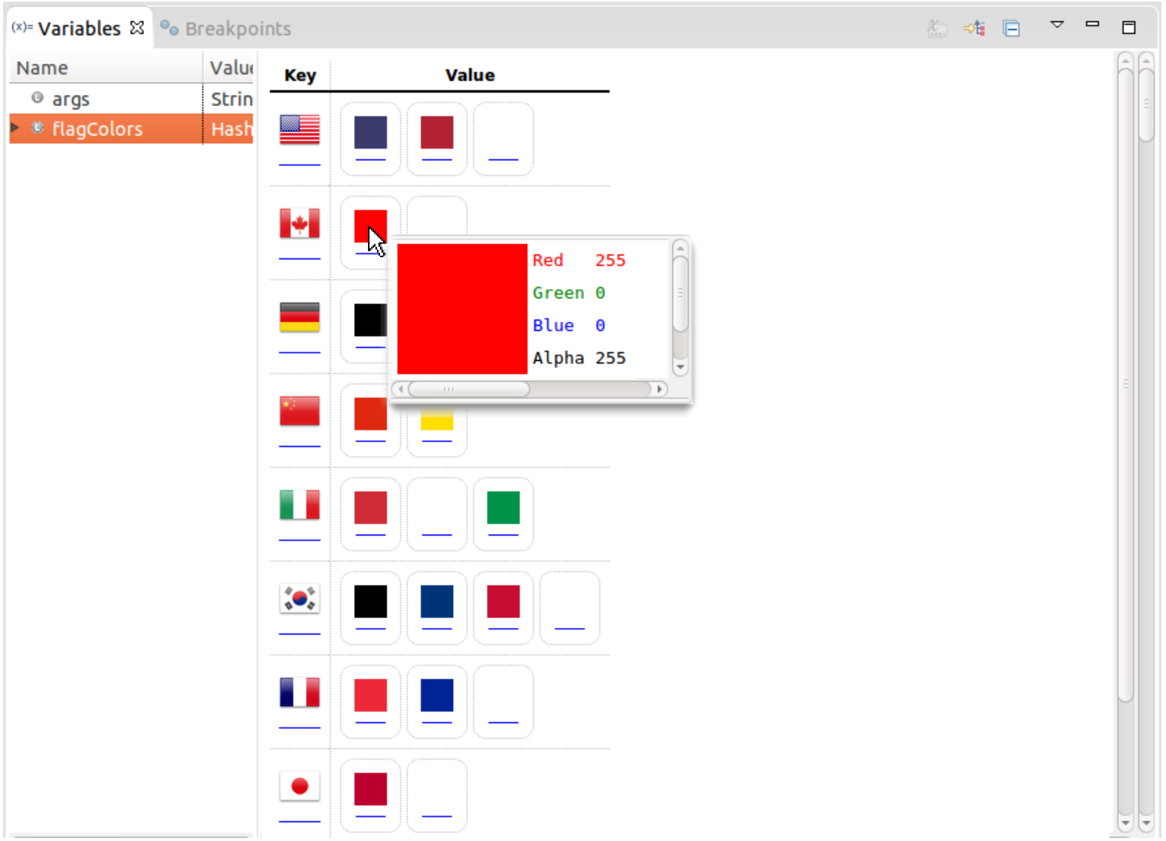
Navigation with the aid of visualizations

Map<Locale, Set<Color>> flagColors = …



Navigation with the aid of visualizations

Map<Locale, Set<Color>> flagColors = …



```
Red    255
Green  0
Blue   0
Alpha  255
```