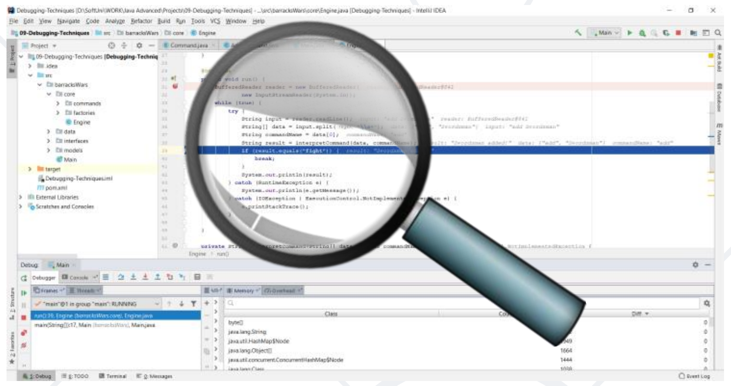
Debugging

Building Rock-Solid Software



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#java-advanced

Table of Contents

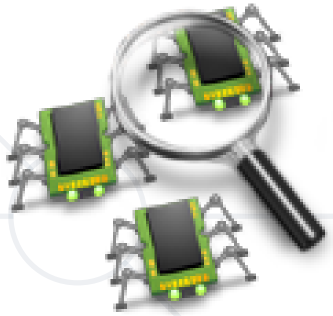
Introduction to Debugging

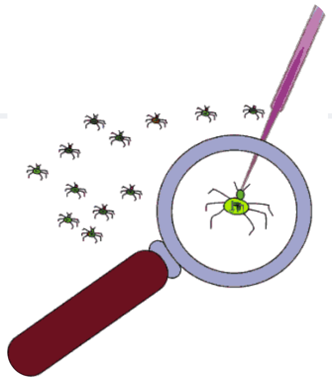
What is Debugging?

- The process of locating and fixing or bypassing **bugs** (errors) in computer program code
- To **debug** a program:
  - Start with a **problem**
  - Isolate the **source** of the problem
  - **Fix** it
- **Debugging tools** (called **debuggers**) help identify coding errors at various development stages

Debugging vs. Testing

- **Testing**
  - A means of initial detection of errors
- **Debugging**
  - A means of diagnosing and correcting the root causes of errors that have already been detected





Importance of Debugging

- $60 Billion per year in economic **losses** due to software **defects**
  - E.g. the **Cluster spacecraft failure** was caused by a bug
- Perfect code is an illusion
  - There are factors that are out of our control
- Legacy code
  - You should be able to debug code that is written years ago
- A deeper understanding of the system as a whole

Debugging Philosophy

- Debugging can be viewed as one big **decision tree**
  - Individual nodes represent **theories**
  - Leaf nodes represent possible **root causes**
  - Traversal of tree boils down to process state **inspection**
  - Minimizing time to resolution is **key**
    - Careful traversal of the decision tree
    - Pattern recognition
    - Visualization and ease of use helps minimize time to resolution

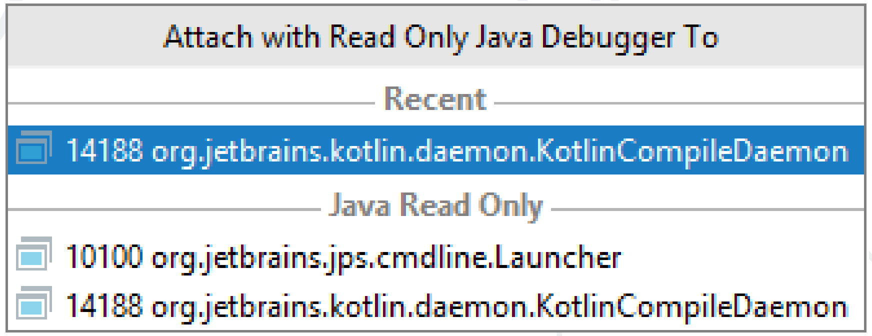
IntelliJ IDEA Debugger

IntelliJ IDEA Debugger

- IntelliJ IDE gives us a lot of **tools** to **debug** your application
  - Adding **breakpoints**
  - Visualize the **program flow**
  - Control the **flow of execution**
  - **Data tips**
  - **Watch variables**
  - Debugging **multithreaded programs**
  - And many more…

How to Debug a Process

- Starting a process under the IntelliJ debugger
- Attaching to an already running process
  - Without a solution loaded you can still debug
  - Useful when a solution isn't readily available
  - **Ctrl** + **Alt** + **F5**



Attach with Read Only Java Debugger To

Recent

14188 org.jetbrains.kotlin.daemon.KotlinCompileDaemon

Java Read Only

10100 org.jetbrains.jps.cmdline.Launcher

14188 org.jetbrains.kotlin.daemon.KotlinCompileDaemon

Debugging a Project

- Right click in **main** method, Debug '{class}.main()'
  - **Shift** + **F9** is a shortcut
- Easier access to the source code and symbols since its loaded in the solution
- Certain differences exist in comparison to debugging an already running process

Debug Windows

- Debug Windows are the means to introspect on the state of a process
- Opens a new window with the selected information in it
- Window categories
  - Frames / Threads
  - Variables
  - Watches
- Accessible from Debug window

Debugging Toolbar

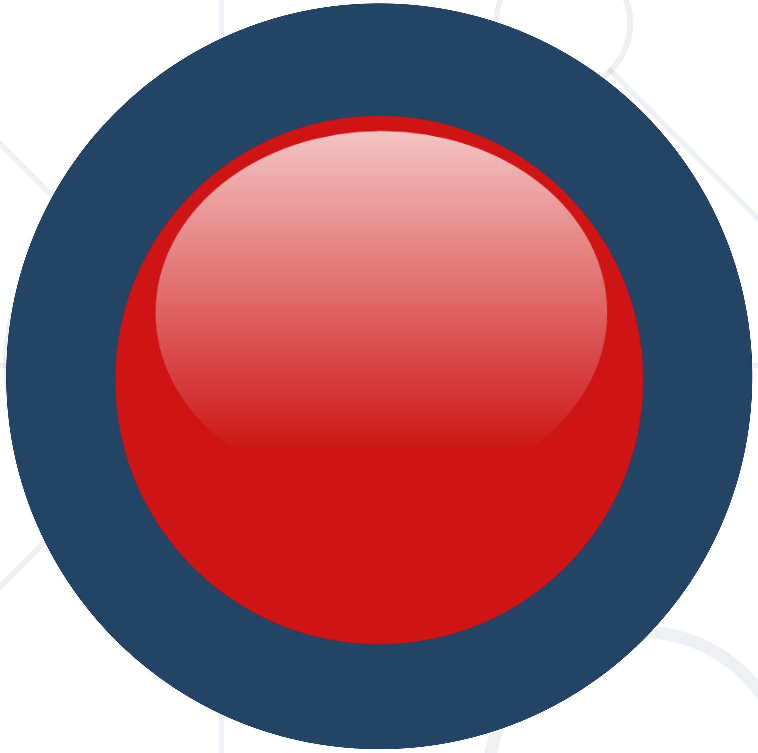
- A convenient shortcut to common debugging tasks
  - **Step over** – F8
  - **Force Step Into** – through the method calls - Alt + Shift + F7
  - **Step Out** – Shift + F8
  - **Step into** – F7
  - **Continue**
  - **Break**
  - **Breakpoints**

Controlling Execution

- By default, an app will run uninterrupted (and stop on exception or breakpoint)
- Debugging is all about looking at the **state of the process**
- Controlling execution allows:
  - **Pausing** execution
  - **Resuming** execution

Options and Settings

- IntelliJ offers quite a few knobs and tweaks in the debugging experience
- Options and settings is available via Settings/Preferences -> Build, Execution and Deployment (Ctrl + Alt + S):
  - Debugger -> Data Views -> **Java**
  - Compiler -> **Java Compiler**
- **Project Structure** (Ctrl + Shift + Alt + S)



Breakpoints

Breakpoints

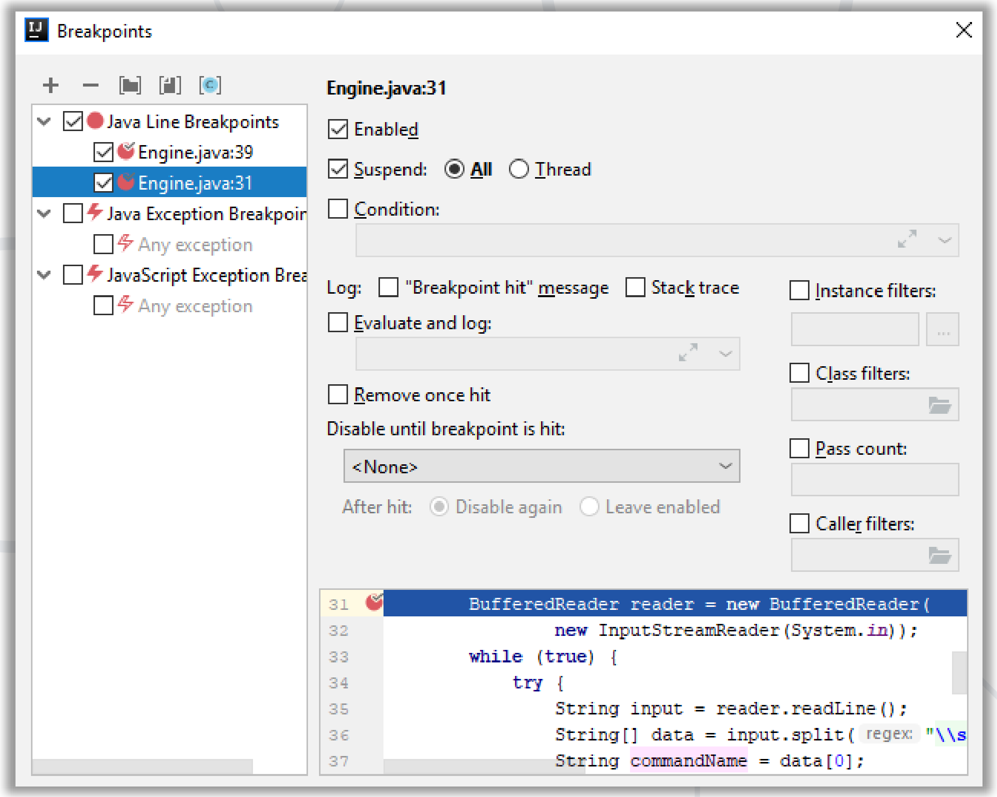
- The ability to stop execution based on certain criteria is key when debugging
  - When a function is hit
  - When data changes
  - When a specific thread hits a function
  - Much more…
- IntelliJ's debugger has a huge feature set when it comes to breakpoints

IntelliJ IDEA Breakpoints

- Stops execution at a specific instruction (line of code)
  - Can be set using:
    - **Ctrl** + **F8** shortcut
    - Clicking on the left most side of the source code window
- By default, the breakpoint will hit every time execution reaches the line of the code
- Additional capabilities: condition, hit count, value changed, when hit, filters

Managing Breakpoints

- Managed in the breakpoint window
- Adding breakpoints
- Removing or **disabling** breakpoints
- Open Breakpoints window
  - Ctrl + Shift + F8



Breakpoints

- [✓] Java Line Breakpoints
  - [✓] Engine.java:39
  - [✓] Engine.java:31
- [ ] Java Exception Breakpoir
  - [ ] Any exception
- [ ] JavaScript Exception Brea
  - [ ] Any exception

Engine.java:31

[✓] Enabled

[✓] Suspend: [✓ All] Thread

[ ] Condition:

Log: [ ] "Breakpoint hit" message [ ] Stack trace

[ ] Evaluate and log:

[ ] Remove once hit

Disable until breakpoint is hit: <None>

After hit: Disable again / Leave enabled

[ ] Instance filters:

[ ] Class filters:

[ ] Pass count:

[ ] Caller filters:

```
        BufferedReader reader = new BufferedReader(
                new InputStreamReader(System.in));
        while (true) {
            try {
                String input = reader.readLine();
                String[] data = input.split( regex: "\\s
                String commandName = data[0];
```

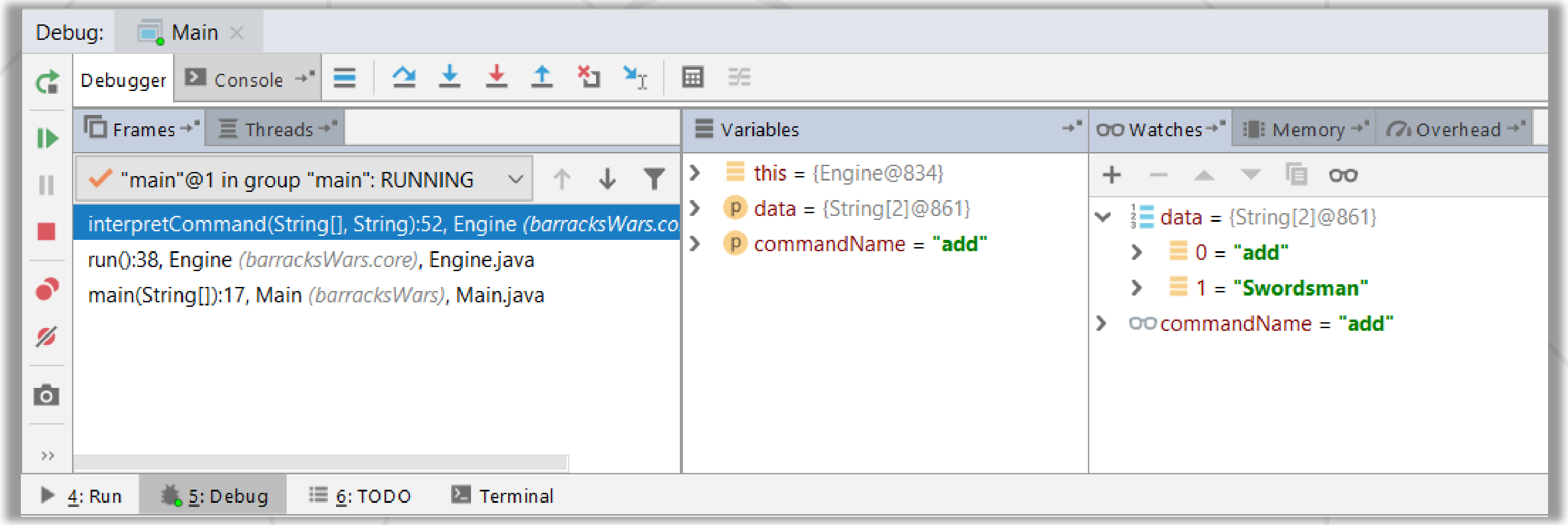
Data Inspection

Data Inspection

- Debugging is all about data inspection
  - What are the **local variables**?
  - What is in **memory**?
  - What is the **code flow**?
  - In general - What is the state of the process right now and how did it get there?
- As such, the ease of data inspection is key to the **quick resolution of problems**

IntelliJ Data Inspection

- IntelliJ offers great data inspection features
  - **Variables**
  - **Watches**
  - **Memory**
  - **Overhead**



```
"main"@1 in group "main": RUNNING
interpretCommand(String[], String):52, Engine (barracksWars.co
run():38, Engine (barracksWars.core), Engine.java
main(String[]):17, Main (barracksWars), Main.java
```

```
this = {Engine@834}
data = {String[2]@861}
commandName = "add"
```

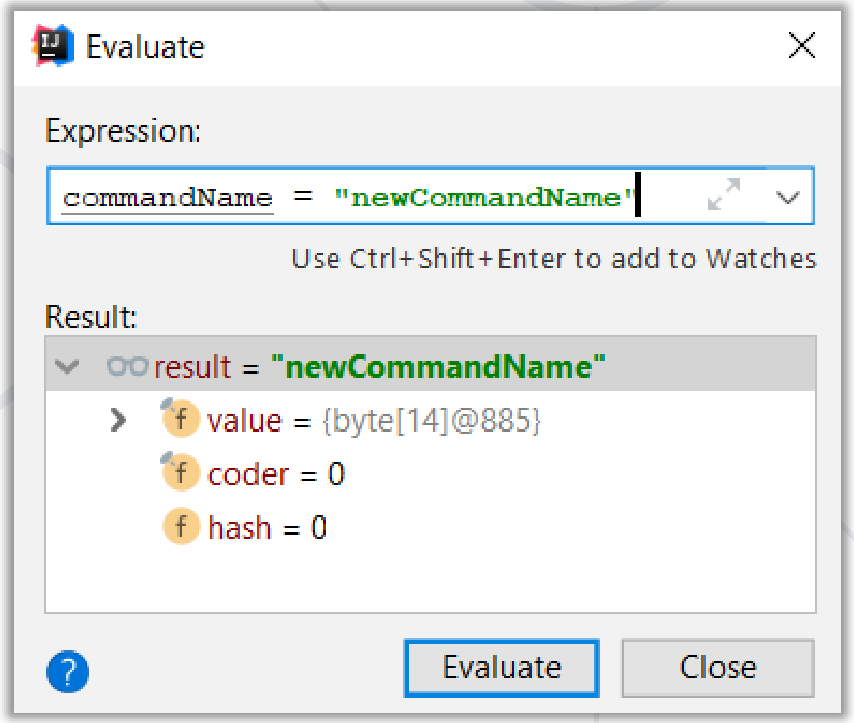
```
data = {String[2]@861}
  0 = "add"
  1 = "Swordsman"
commandName = "add"
```

Variables and Watches Windows

- Allows you to inspect various states of your application
- Several different kinds of "predefined" watches window
- "Custom" watches windows are also possible
  - Contains only variables that you choose to add
  - Right click on the variable and select "Add to Watches"
  - Write the variable name in the Watches window

Evaluate Expression Window

- Enables to evaluate expressions and code fragments in the context of a stack frame
- Also evaluate operator expressions, lambda expressions, and anonymous classes
- Shortcut – Alt + F8



```
commandName = "newCommandName"
```

```
result = "newCommandName"
  value = {byte[14]@885}
  coder = 0
  hash = 0
```

Finding a Defect

Finding a Defect

- Stabilize the error
- Locate the source of the error
  - Gather the data
  - Analyze the data and form a hypothesis
  - Determine how to prove or disprove the hypothesis
- Fix the defect
- Test the fix
- Look for similar errors

Tips for Finding Defects (1)

- Use all available data
- Refine the test cases
- Check unit tests
- Use available tools
- Reproduce the error in several different ways
- Generate more data to generate more hypotheses
- Use the results of negative tests
- Brainstorm for possible hypotheses

Tips for Finding Defects (2)

- Narrow the suspicious region of the code
- Be suspicious of classes and routines that have had defects before
- Check code that's changed recently
- Expand the suspicious region of the code
- Integrate incrementally
- Check for common defects
- Talk to someone else about the problem
- Take a break from the problem

Fixing a Defect

- Understand the problem before you fix it
- Understand the program, not just the problem
- Confirm the defect diagnosis
- Relax
- Save the original source code
- Fix the problem, not the symptom
- Make one change at a time
- Add a unit test that exposes the defect
- Look for similar defects

Source: http://www.movingseniorsbc.com

Psychological Considerations

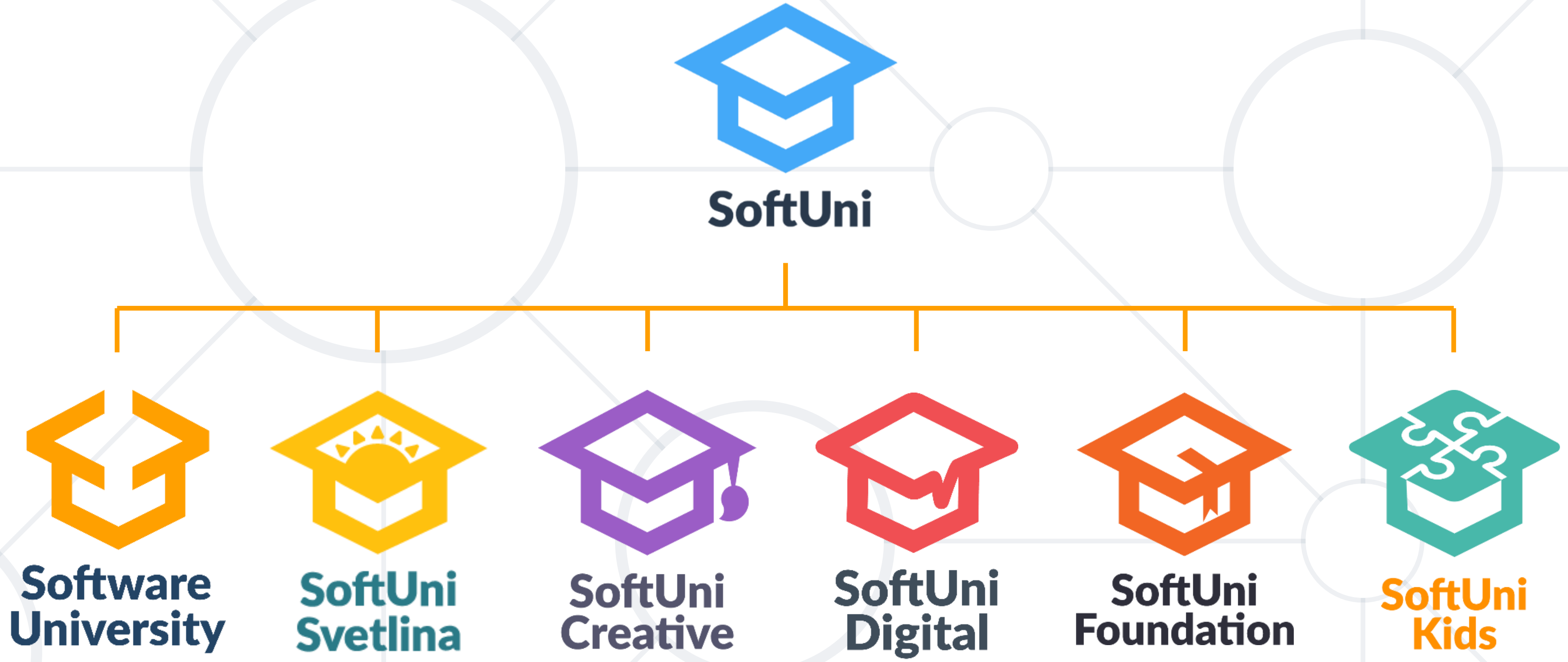
- Your ego tells you that your code is good and doesn't have a defect even when you've seen that it has
- How "psychological set" contributes to debugging blindness
  - People expect a new phenomenon to resemble similar phenomena they've seen before
  - Do not expect anything to work "by default"
  - Do not be too devoted to your code – establish psychological distance

Summary

- Introduction to **Debugging**
- IntelliJ IDEA Debugger
- **Breakpoints**
- Data Inspection
  - **Variables, Watches, Frames**
- Finding a **Defect**

Software University

Questions?



SoftUni Diamond Partners

Software University

DraftKings

POKERSTARS | CAREERS

SCHWARZ

LIDL

Kaufland

SCHWARZ Produktion

prezero

Postbank

Решения за твоето утре

SOFTWARE GROUP

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

SmartIT

INDEAVR

Serving the high achievers

Bosch.IO

PHAR VISION

createX

taulia®

Educational Partners

CODE IT UP

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

FORUM

License