PV 168
Seminar 13

Prerequisites for the seminar

- **NetBeans IDE 12.0**
  - Download from https://netbeans.apache.org/download/index.html

Address Database

- Loads list of addresses in Czech Republic (`DataLoader`)
- Find all addresses matching given (possibly incomplete) specification (`AddressFinder`)
- Executes performance test to help evaluate CPU and Memory consumption (`PerformanceTest`)
- There are multiple implementations of AddressFinder using various data structures and search algorithms
- Concrete implementation is selected with dialog box when the application is started

`SimpleAddressFinder`

- Stores data as simple List, no optimized structure
- Search is done sequentially, all addresses must be traversed
- Multiple search strategies:
  - **`ForEachSearchStrategy`** – based on for-each loop (**#1**)
  - **`StreamSearchStrategy`** – based on Streams (**#2**)
  - **`ParallelStreamSearchStrategy`** – based on parallel Streams. Parallel streams utilize multiple threads with Fork/Join framework. Threads count corresponds to CPU count. (**#3**)

`IndexedAddressFinder`

- Stores data in map-based structure.
- The search process consist of two steps
  - Finding the collection of addresses with appropriate **`AddressBase`** (*municipality*, *municipality district*, *street*, and *district*).
  - Then finding addresses within this collection with appropriate *orientation number* and/or *house number*.
- The first step is implemented as a map lookup to avoid sequential search.
- Multiple implementations exist for the second step:
  - **`IndexedAddressGroup`** – addresses with the same **`AddressBase`** are stored in Maps, find by number(s) is done as map lookup (**#4**)
  - **`SimpleAddressGroup`** – addresses with the same **`AddressBase`** are stored in simple List, find by number(s) is done sequentially (**#5**)

Profiler

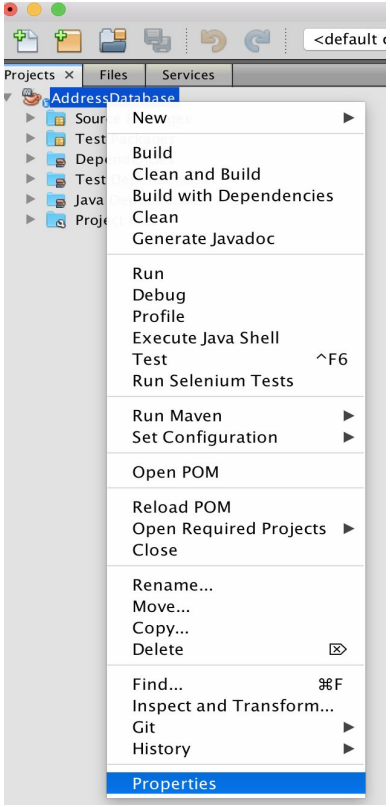
- Tools for evaluation of application performance
  - CPU time
  - Memory usage
- Profiler integrated in IntelliJ IDEA does not provide good support for memory usage profiling 😕

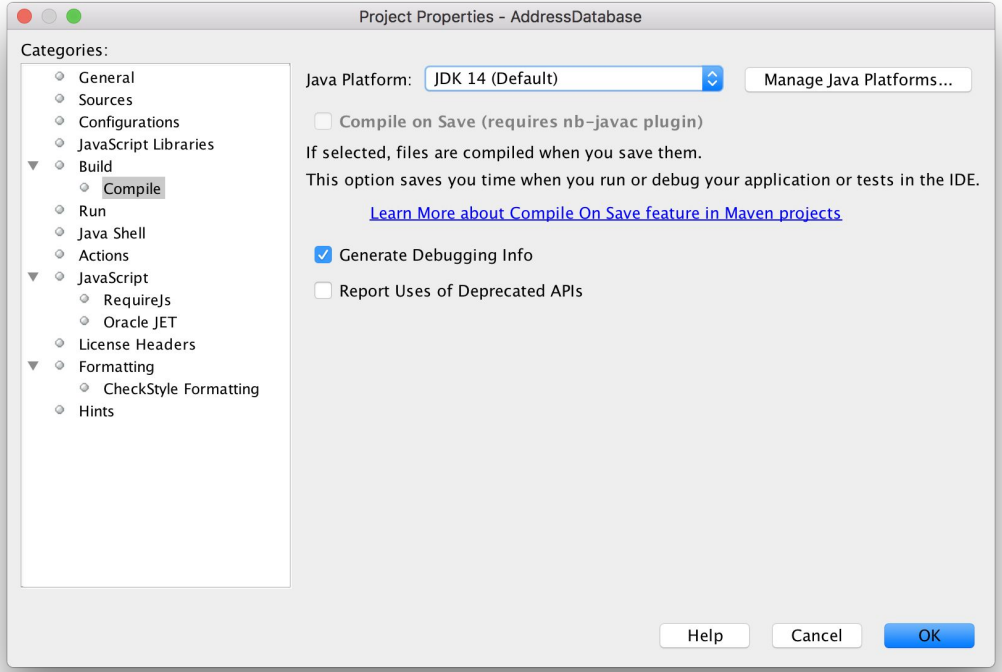
Instructions (screenshots on next slides)

- Clone the project https://gitlab.fi.muni.cz/pv168/address-database
- Open the project in NetBeans
- Set Java Platform to JDK 11 or newer
  - Right click on project AddressDatabse in left panel and choose *Properties*
  - Select category *Build* > *Compile* and choose Java Platform
  - If you don't see suitable JDK there, click on Manage Java Platforms… to add JDK
- Open profiler (menu *Profile* > *Profile Project* or *Ctrl+F2*)
- Configure Sesion by selecting profile Telemetry
- Run the application in profiler 

- Choose AddressFinder implementation **#1**
- Check the results (notice the detail numbers when hovering over the graphs)

How to set JDK in NetBeans





Profiler window

AddressDatabase | pom.xml

Configure Session

Configure and Start Profiling

1. Click the **Configure Session** button in toolbar and select the desired profiling mode:

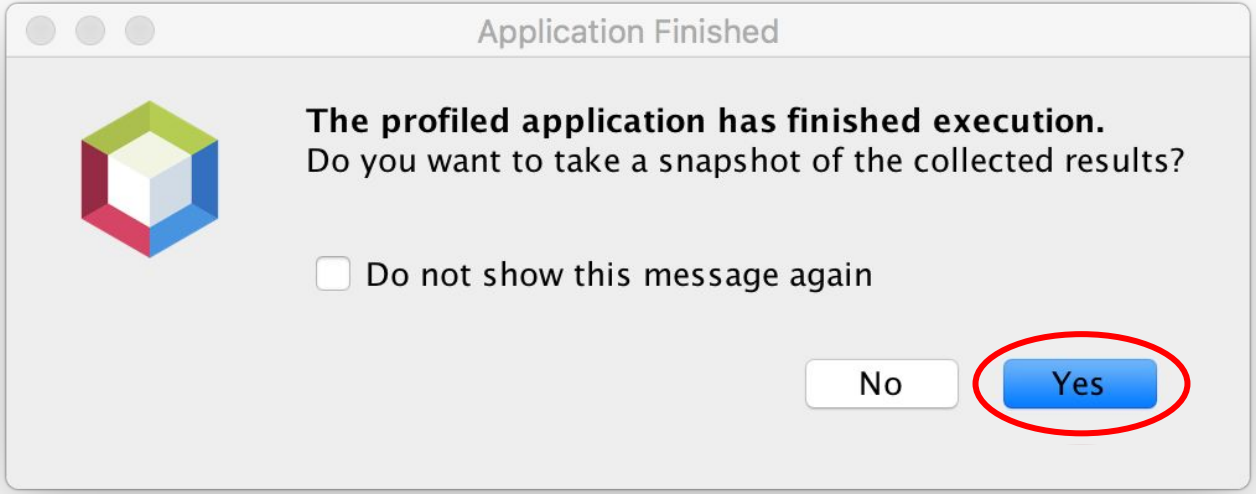
Telemetry	Monitor CPU and Memory usage, number of threads and loaded classes
Methods	Profile method execution times and invocation counts, including call trees
Objects	Profile size and count of allocated objects, including allocation paths
Threads	Monitor thread states and times
Locks	Collect lock contention statistics
SQL Queries	Display executed SQL queries, their duration and invocation paths

2. Click the **Profile** button in toolbar once the session is configured to start profiling.
3. Use the Profile **dropdown arrow** to change profiling settings for the session.

Tips & Tricks: Snapshots

- Methods and Objects profiles allow to take snapshots of the results
  - Useful for later comparison results for different **`AddressFinder`** implementations
  - Unfortunately there is no such option for Telemetry
- Snapshot is taken if you confirm it in dialog when the application finishes
- Snapshots can be saved to disk
  - For long term comparison
- Saved snapshots can be renamed
  - In window Snapshots (menu *Window* > *Profiling* > *Snapshots*)
  - Better identification, e.g. AddressFinder #1

Tips & Tricks: Taking Snapshot



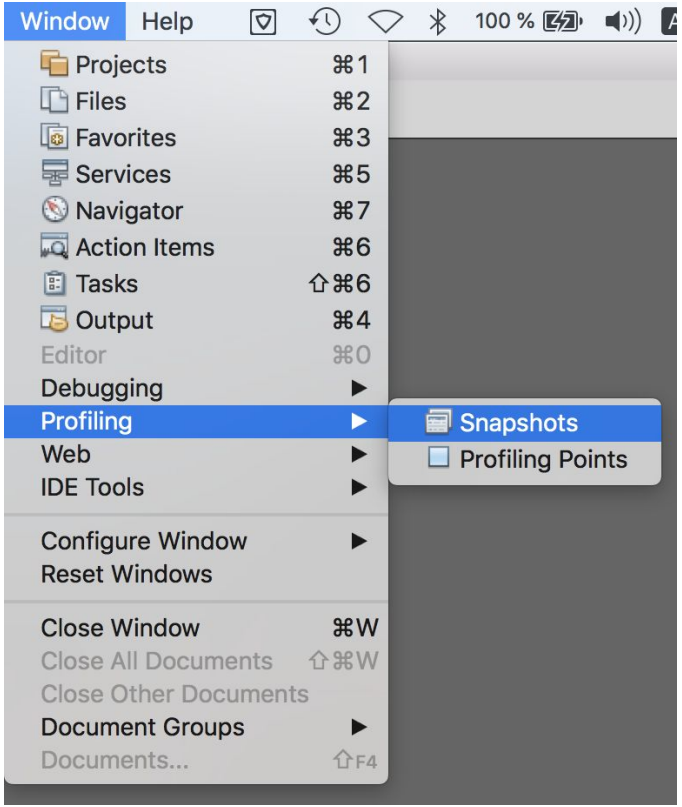
Tips & Tricks: Saving Snapshot

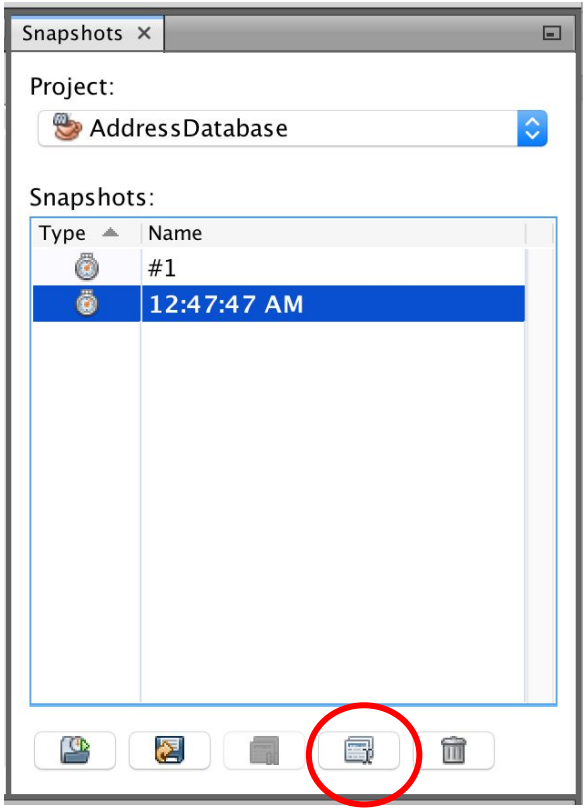
AddressDatabase | pom.xml × 12:47:47 AM * ×

View: Aggregation: Methods

Name	Total Time		Total Time (CPU)	
Java2D Queue Flusher	19,953 ms	(100%)	123 ms	(100%)
main	19,531 ms	(100%)	14,644 ms	(100%)
Finalizer	9,690 ms	(100%)	10.9 ms	(100%)
AWT-Shutdown	7,525 ms	(100%)	0.0 ms	(-%)
AWT-EventQueue-0	7,146 ms	(100%)	3,850 ms	(100%)
Common-Cleaner	5,476 ms	(100%)	12.7 ms	(100%)
Reference Handler	5,463 ms	(100%)	5,463 ms	(100%)
AWT-Shutdown	2,965 ms	(100%)	0.0 ms	(-%)
AWT-Shutdown	2,675 ms	(100%)	0.0 ms	(-%)
AWT-Shutdown	1,637 ms	(100%)	0.0 ms	(-%)
AWT-Shutdown	1,380 ms	(100%)	0.0 ms	(-%)
TimerQueue	1,333 ms	(100%)	0.0 ms	(-%)
AppKit Thread	680 ms	(100%)	657 ms	(100%)
process reaper	0.0 ms	(-%)	0.0 ms	(-%)
Java2D Disposer	0.0 ms	(-%)	0.0 ms	(-%)
DestroyJavaVM	0.0 ms	(-%)	0.0 ms	(-%)
AWT-Shutdown	0.0 ms	(-%)	0.0 ms	(-%)

Tips & Tricks: Renaming Saved Snapshot





Seminar Task 1 (AddressFinder **#1**)

- Run the application with ***Telemetry*** profile
  - Check *Memory* graph to see *used heap size* after loading the data.
  - Check the output tab to see average time per single search
  - Write down both numbers
- Run the application with ***Methods*** profile
  - Check where the application spent most of the time in ***main*** thread
- Run the application with ***Objects*** profile
  - Check which object types occupied most of the heap

Seminar Task 2 (AddressFinder **#2**)

- Run the application with ***Telemetry*** profile
  - Check *Memory* graph to see *used heap size* after loading the data.
  - Check the output tab to see average time per single search
  - Write down both numbers
- Run the application with ***Methods*** profile
  - Check where the application spent most of the time in ***main*** thread
- Run the application with ***Objects*** profile
  - Check which object types occupied most of the heap
- Discuss and write down answers to these questions:
  - Is there any significant difference in CPU time or memory consumption compared to **#1**?

Seminar Task 3 (AddressFinder **#3**)

- Run the application with ***Telemetry*** profile
  - Check *Memory* graph to see *used heap size* after loading the data.
  - Check the output tab to see average time per single search and **CPU count**
  - Write down all three numbers
- Run the application with ***Methods*** profile
  - Check where the application spent most of the time in ***main*** thread
  - Check where the application spent most of the time in ***ForkJoinPool.**** threads
- Run the application with ***Objects*** profile
  - Check which object types occupied most of the heap
- Discuss and write down answers to these questions:
  - Is there any significant difference in CPU time or memory consumption compared to **#1** or **#2**?
  - How many ***ForkJoinPool.**** threads were running?

Seminar Task 4 (AddressFinder **#4**)

- Run the application with ***Telemetry*** profile
  - Check *Memory* graph to see *used heap size* after loading the data.
  - Check the output tab to see average time per single search
  - Write down both numbers
- Run the application with ***Methods*** profile
  - Check where the application spent most of the time in ***main*** thread
- Run the application with ***Objects*** profile
  - Check which object types occupied most of the heap
- Discuss and write down answers to these questions:
  - Is there any significant difference in CPU time or memory consumption compared to **#1** – **#3**?

Seminar Task 5 (AddressFinder **#5**)

- Run the application with ***Telemetry*** profile
  - Check *Memory* graph to see *used heap size* after loading the data.
  - Check the output tab to see average time per single search
  - Write down both numbers
- Run the application with ***Methods*** profile
  - Check where the application spent most of the time in ***main*** thread
- Run the application with ***Objects*** profile
  - Check which object types occupied most of the heap
- Discuss and write down answers to these questions:
  - Is there any significant difference in CPU time or memory consumption compared to **#1** – **#4**?

Seminar Task 6 (Evaluation)

- Which implementation was the least CPU efficient (slowest) one?
- Which implementation was the most CPU efficient (fastest) one?
- How much did parallel processing in #3 help?
- What is the cost of optimization in #4? Is it worth it?
- Which implementation would you recommend to use?

Link to slides

https://is.muni.cz/auth/el/fi/podzim2020/PV168/um/seminare/PV168-seminar-13.pdf

Conclusion

Any questions?