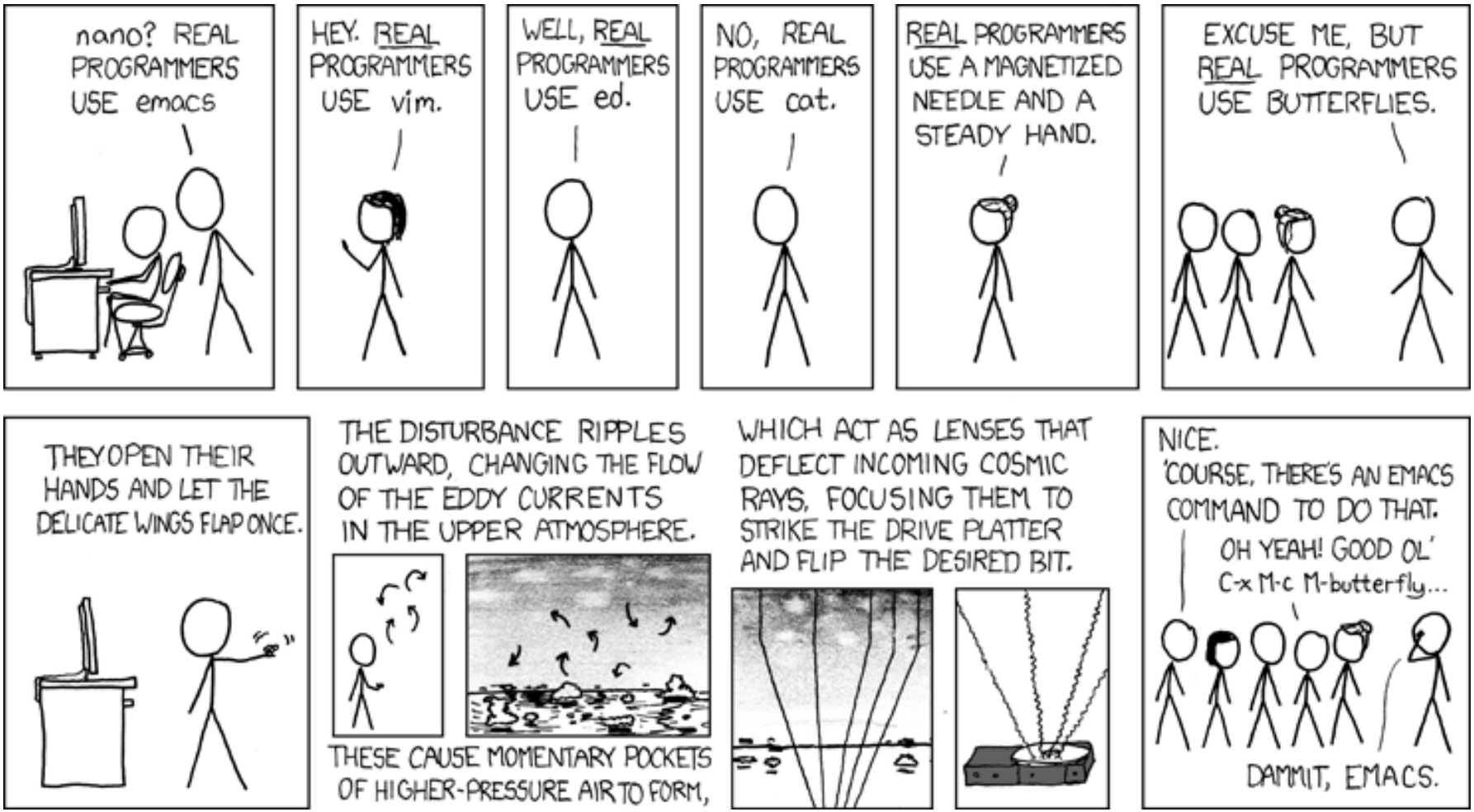
“A Magnetized Needle and a Steady Hand”

Alternatives in the modern world of Integrated Development Environments

Jennifer Wood
CSCI 5828
Spring 2012

Real Programmers



http://xkcd.com/378/

For the rest of us

- Modern Integrated Development Environments (IDE)
  - A one-stop shop with multiple features that can be easily accessed by the developer (without switching modes or activating other utilities) to ease the task of creating software
  - A multitude of IDEs exist for each programming language (Java, C++, Python, etc.) and each platform (desktops, cell phones, web-based, etc.)
  - Some IDEs can handle multiple programming languages, but most are based in just one
  - There are many good free IDEs out there, but you can also pay for functionality from $ to $$$$
  - IDEs are like opinions, everyone has one and everyone thinks everyone else's stinks

Why are IDEs a good thing?

- They attack many of the sources of accidental difficulties in software development by having:
  - Real-time protection from fault generating typos and bad syntax
  - High levels of abstraction to keep developers from being forced to redevelop basic (and not so basic) classes and structures for every project
  - IDE increases the power of many development tools by merging them into one that provides “integrated libraries, unified file formats, and pipes and filters. As a result, conceptual structures that in principle could always call, feed, and use one another can indeed easily do so in practice.” (Brooks, 1987).
    - A core focus of IDE developers is continuous improvement in transparency to minimize searching for functions or switching modes by the user

Why are Modern IDEs a Good Thing?

- The original IDEs were command driven and provided minimal assistance in the development of source code
  - Huge potential for making errors when coding
  - Resulted in large amounts of time spent debugging code
    - Often the debugging tools were located in a separate tool that had to be booted up by the user and run with the problem piece of code
  - Little support with the language itself to help avoid typos or bad syntax
  - You could argue that at this point accidental difficulties in software development were still running rampant

Common IDE Features

- These basics are generally included:
  - Text editor – allows writing and editing of source code
  - Build automation – generates the Makefile (or equivalent) and passes it to the compiler
  - Debugger – allows the developer to follow the execution of source code step by step to confirm correct execution or to locate faults
- These additional features are common:
  - Compiler/Interpreter
  - Error checking as you write the source code
  - Code autocompletion as you type
  - Highlighting of code syntax (including bracket matching))
  - Ability to browse and/or visualize class structures and hierarchies
  - Templates for quick creation of classes

Nice-to-Have Features

- Profiler – monitors execution of software to aid in discovery of memory leaks and performance improvement
  - Task profiling
  - Heap monitoring – look for unnecessary references and sources of memory leaks
  - Thread monitoring – check status of multiple threads
  - Identify CPU bottlenecks – methods that are monopolizing large amounts of processing time
- Database Integration – supports database development and usage
  - Often multiple database frameworks are supported (SQL, Oracle DB, Apache Derby, etc.)
- Storage Management – handles the saving and loading of multiple types of data
- Version control – keep track of what is being edited and which version is the most up to date
- UML Support – design your system with UML diagrams and use these blueprints within your IDE to build the supporting code

Nice-to-Have Features (2)

- GUI Design Tool – drag and drop design of GUI with the IDE autocoding much of the GUI framework in the source code with placeholders for user supplied methods
- Refactoring Support – assists in safe deletion of classes, encapsulation of fields (adding get/set methods), breaking a class apart into a new abstract class and subclass, renaming classes, etc.
- Testing Support
  - Code coverage
  - Unit Testing
  - Test case management
- Issue Tracking – support for maintaining a database of open issues that can be shared with the development team
- Lifecycle Integration – support for Agile tools and planning

IDEs for tomorrow, today

- Support for programs cobbled together with multiple programming languages
  - Allows developers to leverage the strengths of multiple programming languages when solving and modeling complex problems
- IDEs are available that can be run in a browser with your source code stored in the cloud
  - Supports flexible development environments where teams can share and access resources from multiple computers and locations
- IDEs are evolving to support simultaneous development for multiple frameworks and/or devices
  - Example: Your company wants you to build a new game such that can be released for desktops, tablet computers, and cell phones

Some drawbacks with modern IDEs

- Modern IDEs are complex!
  - To fully utilize the features of a big IDE takes an investment of time and training
  - There can be a lot of clutter as all the features clamor for space in the IDE workspace
  - They take up a lot more disk space and use a lot more memory than just running Vim, GCC and GDB
  - New revisions roll out regularly and features you liked can be moved or may no longer be supported
    - Example: NetBeans 7.0 has stopped supporting Ruby and Ruby on Rails (but you can get a third party plug-in)

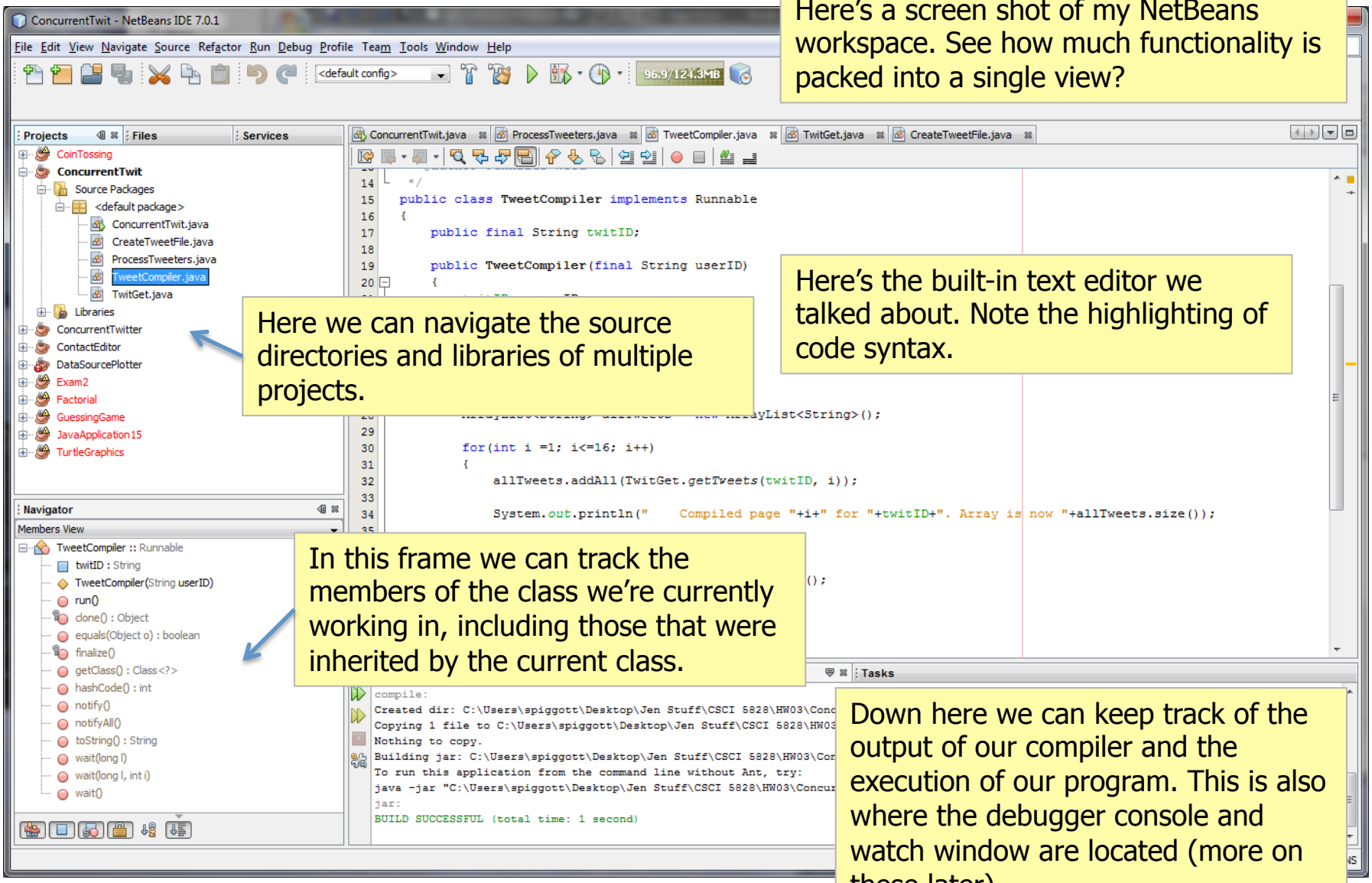
Finding the Right IDE for Your Project (and You)

- Cost
  - There are many good free IDEs out there (Apple Xcode, Eclipse, NetBeans, etc.)
  - There are also many good IDEs that require a paid license (Microsoft Visual Studio, IBM Rational Rhapsody, etc.)
- Functionality
  - Availability of plug-ins or modules to expand capabilities
  - Multi-language support (many IDEs support a single programming language)
  - Is the platform or device you are designing for supported by the IDE?
  - Are there development team/product lifecycle supporting features?
  - Wikipedia has a nice set of basic comparison tables for many types of IDEs: http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments
- Comfort
  - Go with what works for you
    - Get input from the people you work with and/or people in the larger community that you respect
      - Good resources for IDE selection can be places like slashdot.org, stackoverflow.com, etc.
- Reality
  - What will your company's IT department actually let you install on your machine?

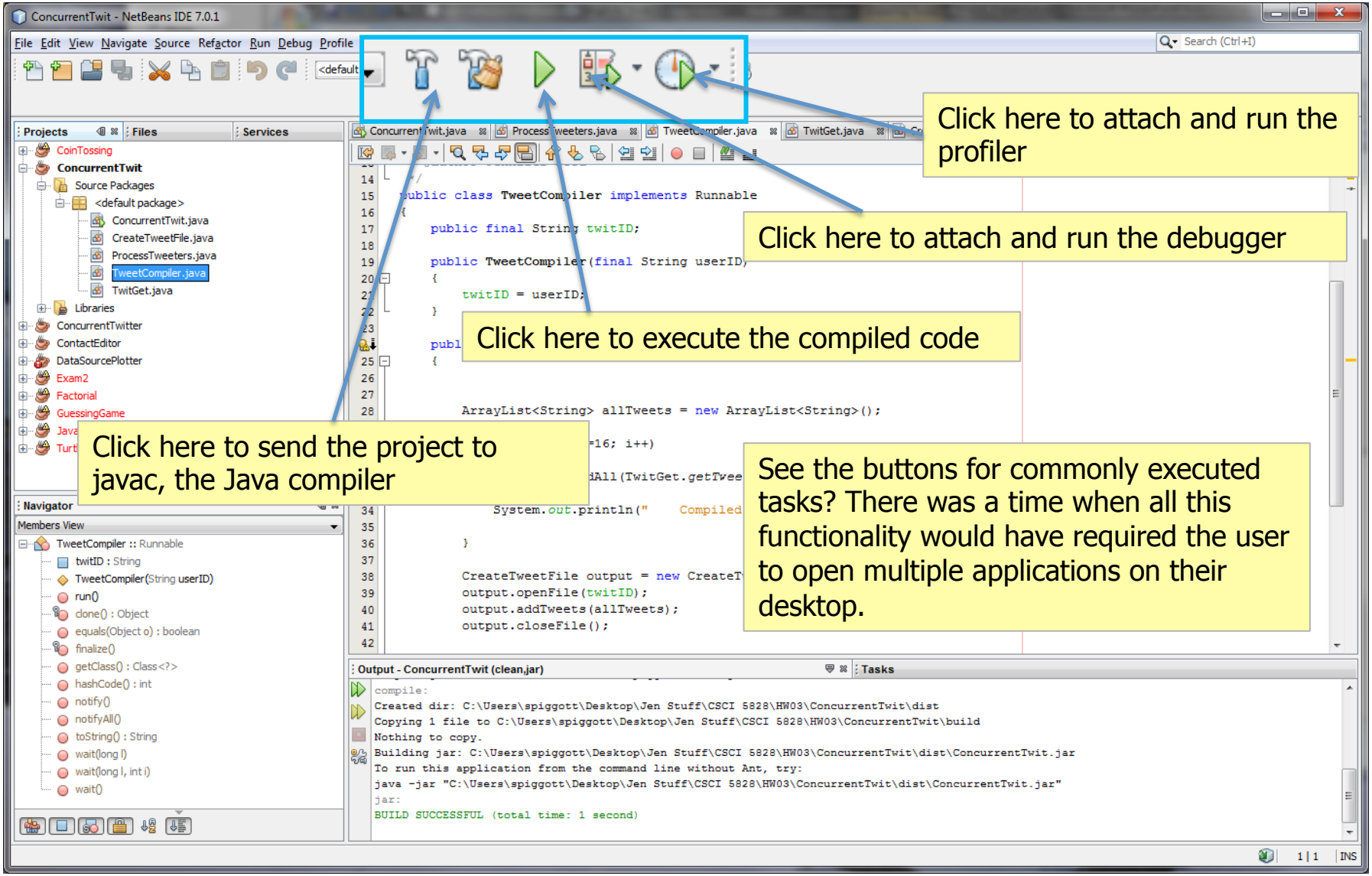
Looking at NetBeans:
An Example of a Full-Featured IDE

- Supports Windows, Mac, Linux and Solaris
- Design and build web, desktop and mobile applications in Java
- Multilanguage support for Java, PHP, JavaScript, Ajax, Groovy and Grails, and C/C++
  - Ruby/Ruby on Rails no longer directly supported, but a plug-in is available at http://plugins.netbeans.org/plugin/38549/ruby-and-rails
- Includes some nice-to-have features:
  - Drag-and-drop GUI Design Tool (now with Visual Debugging!)
  - Supports Java Enterprise and web application development for multiple frameworks
  - Database Explorer to connect to, search, create or delete any relational database for which there is a JDBC driver
  - Version Control with Concurrent Versions System (CVS) including version integration
  - Agile friendly tools like Team Project hosting, issue tracking, and continuous build support

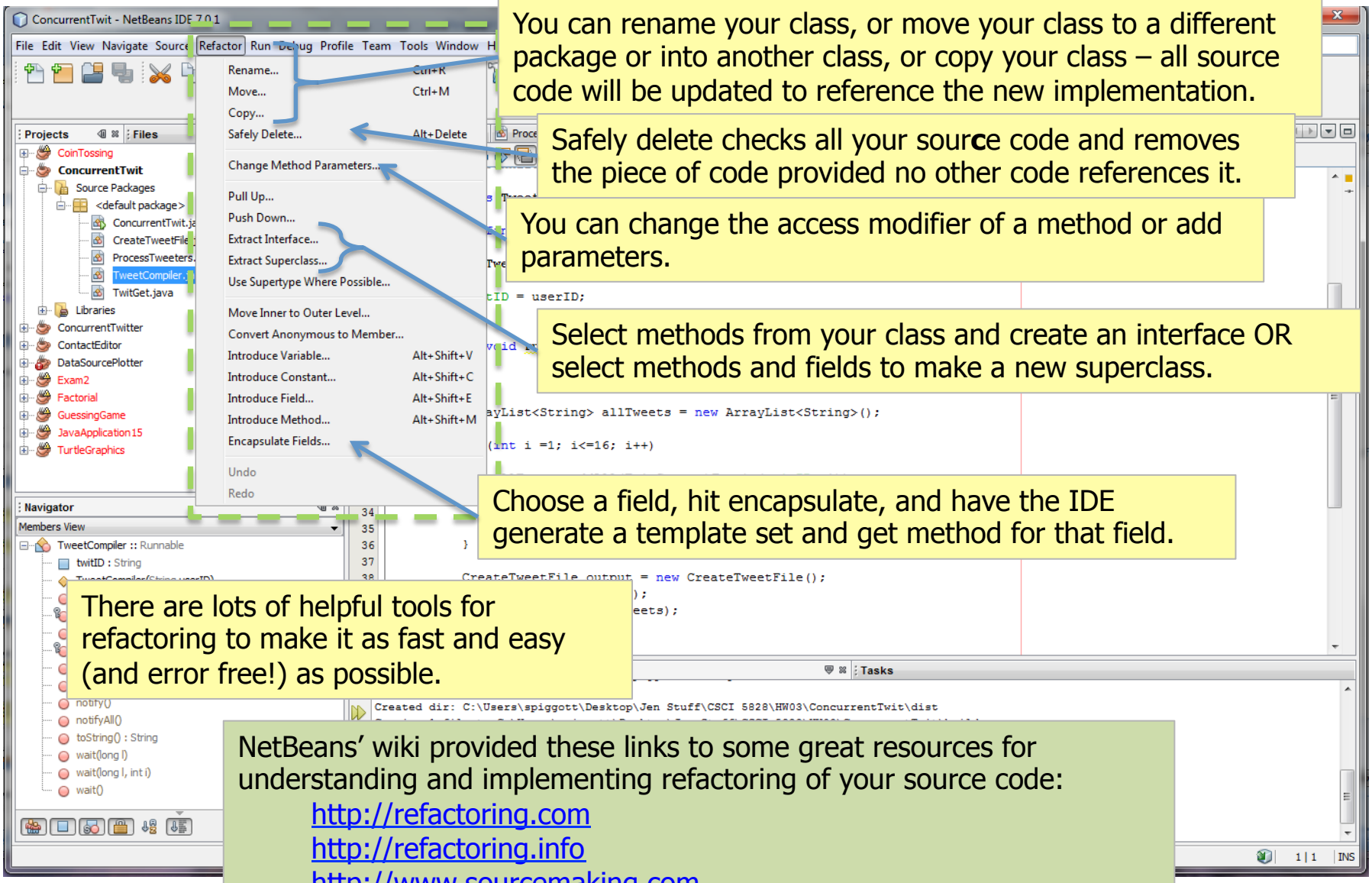
NetBeans – A Brief Tour: The Workspace



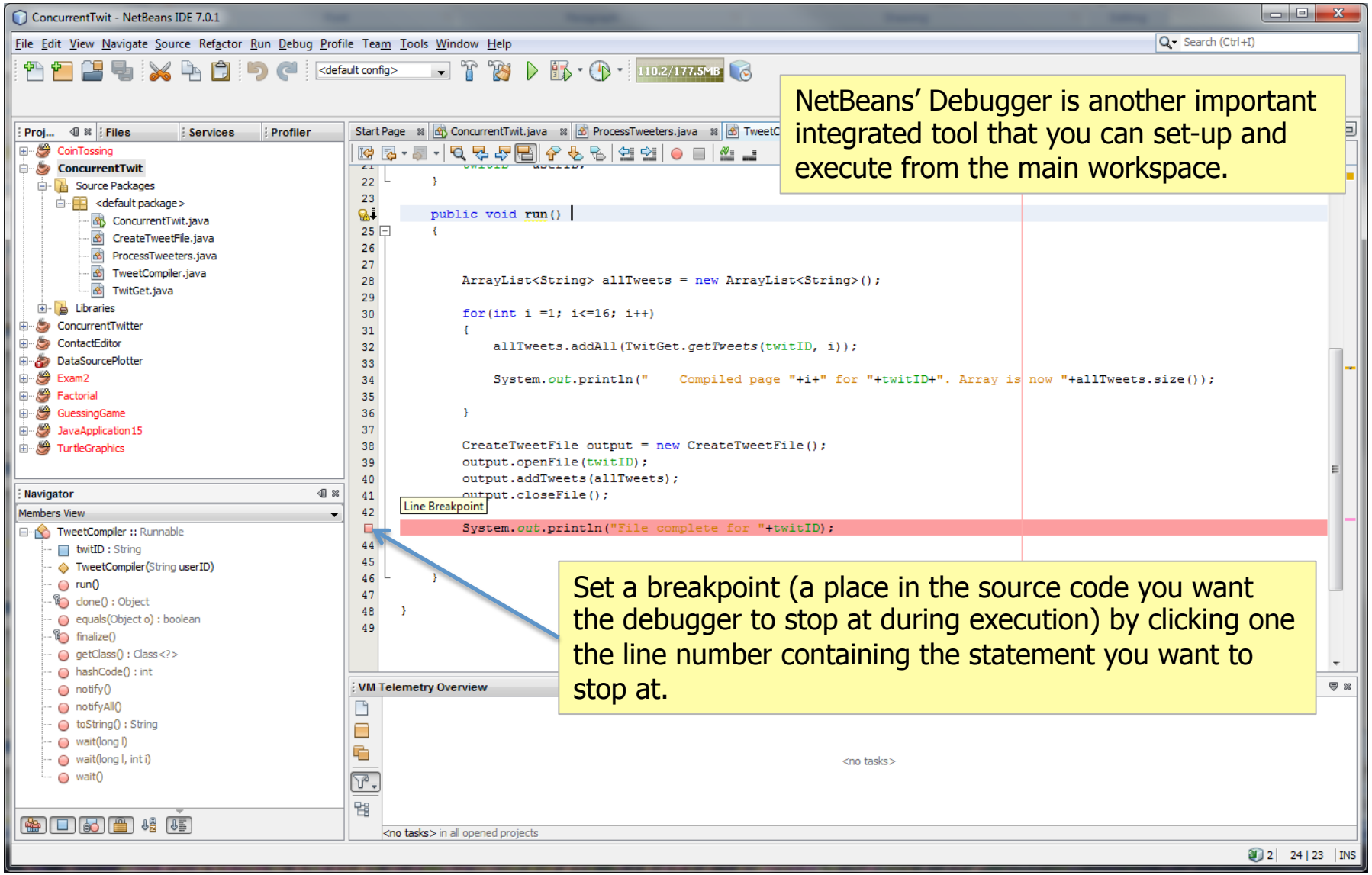
NetBeans – A Brief Tour: The Workspace (2)



NetBeans – A Brief Tour: Refactoring



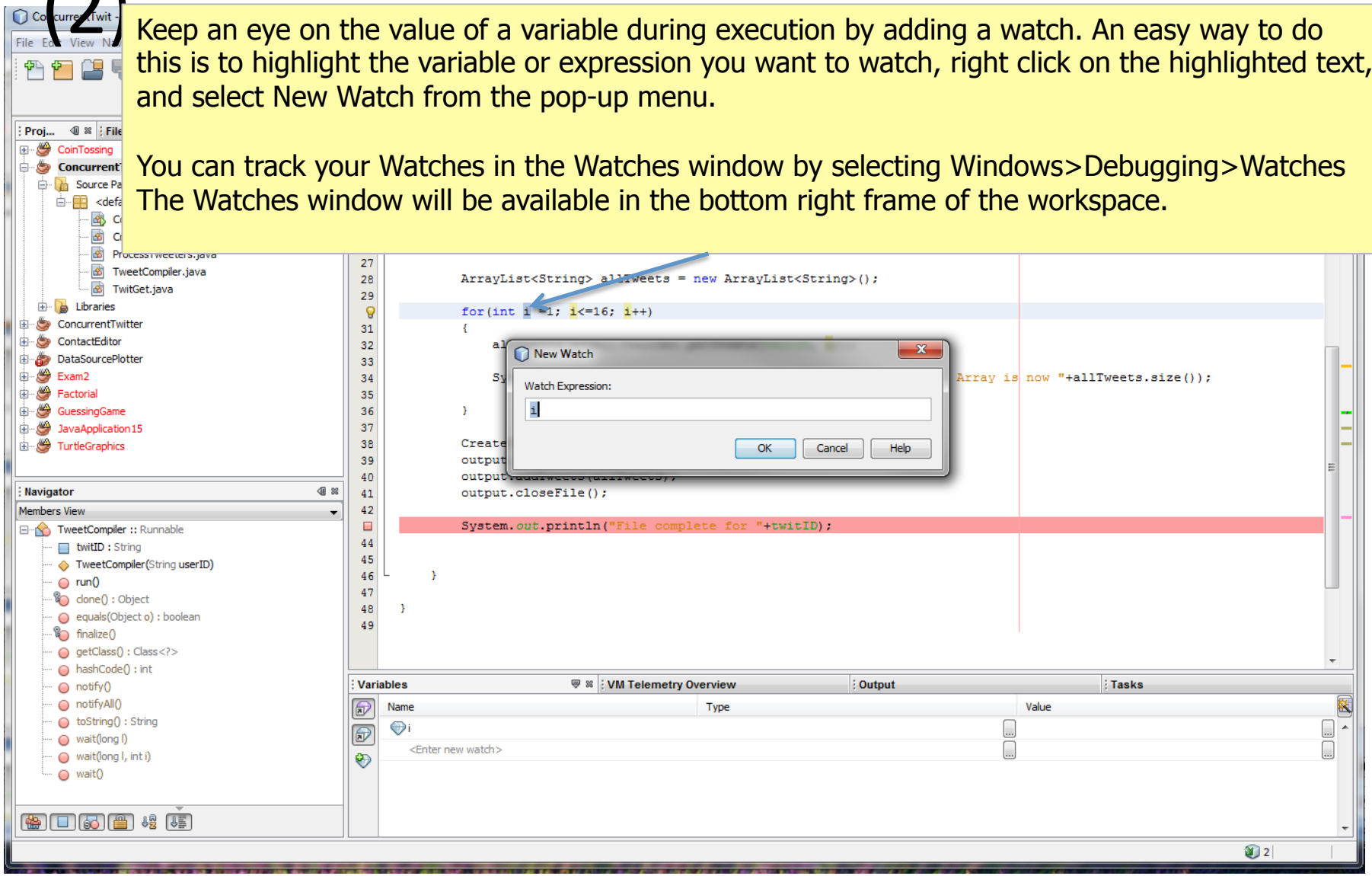
NetBeans – A Brief Tour: Debugging



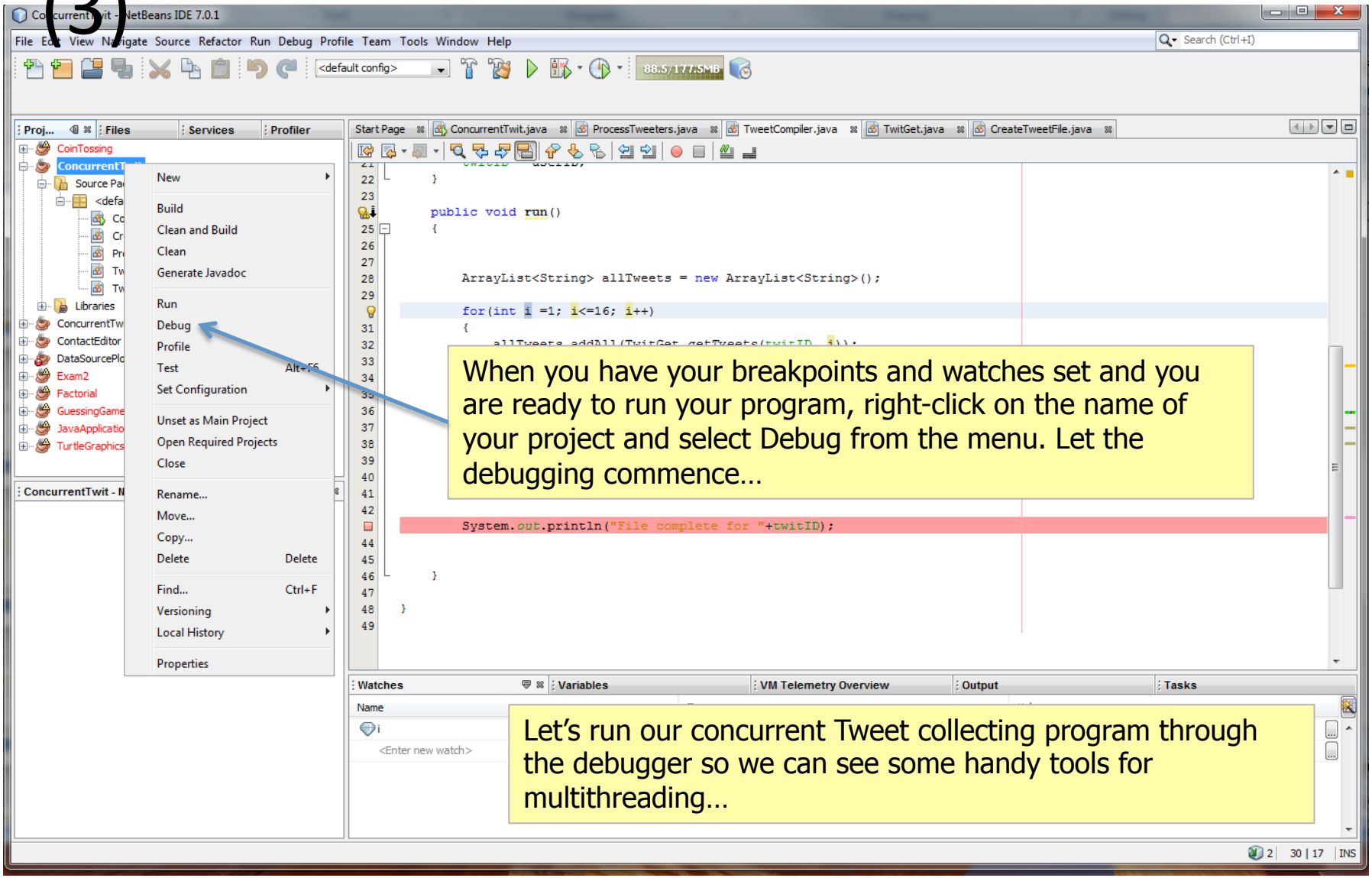
NetBeans – A Brief Tour: Debugging (2)

Keep an eye on the value of a variable during execution by adding a watch. An easy way to do this is to highlight the variable or expression you want to watch, right click on the highlighted text, and select New Watch from the pop-up menu.

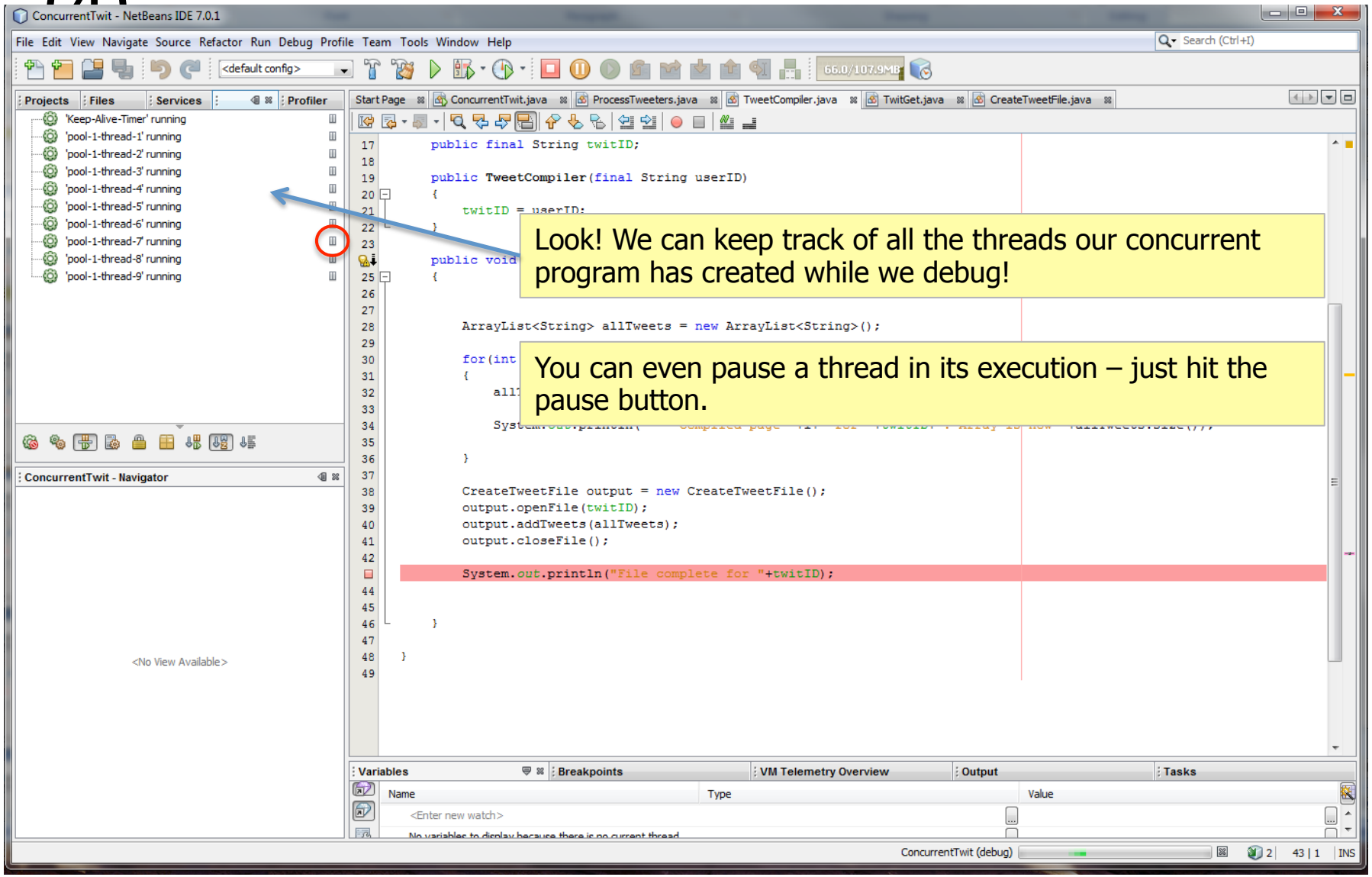
You can track your Watches in the Watches window by selecting Windows>Debugging>Watches The Watches window will be available in the bottom right frame of the workspace.



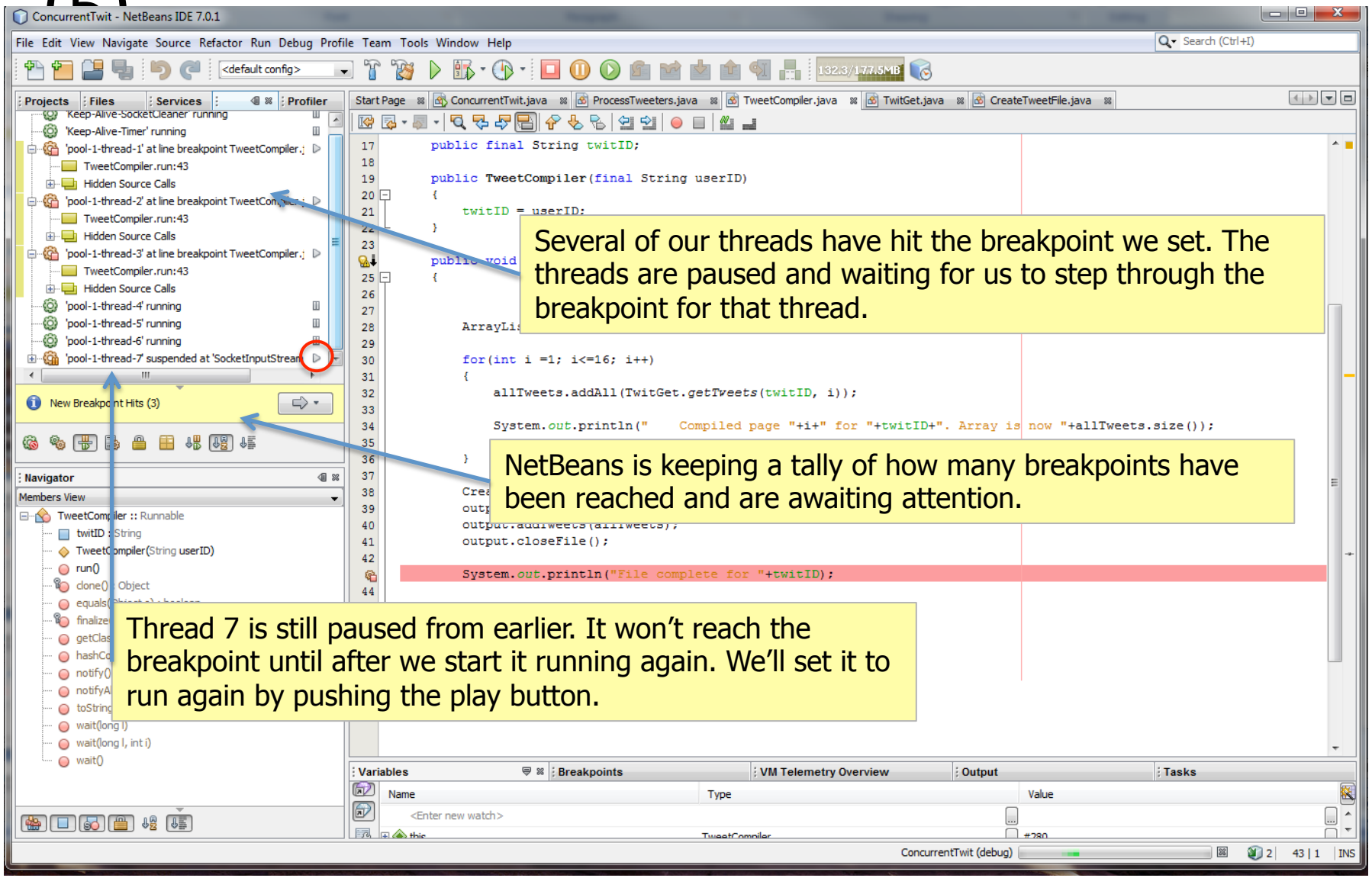
NetBeans – A Brief Tour: Debugging (3)



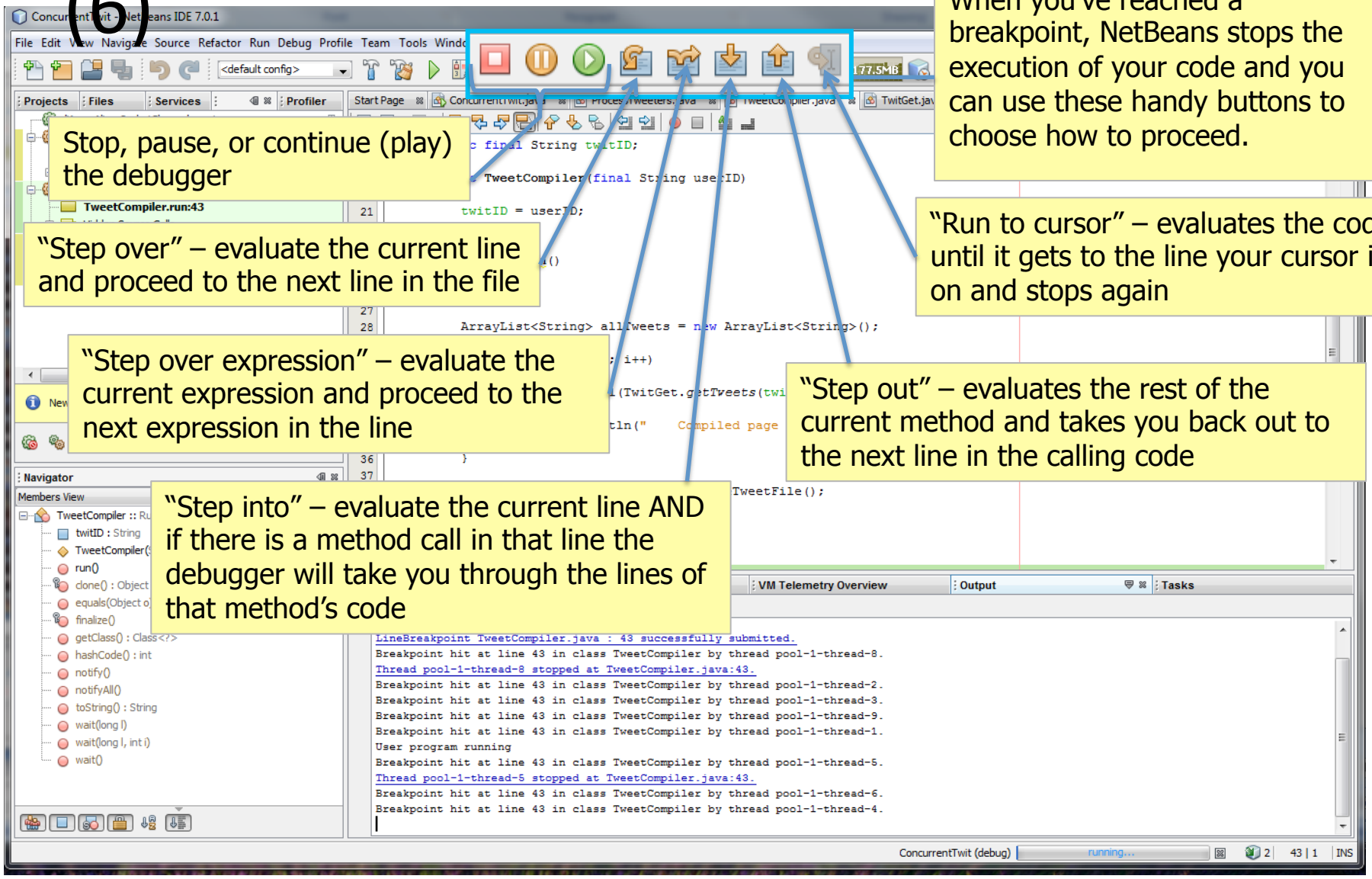
NetBeans – A Brief Tour: Debugging (4)



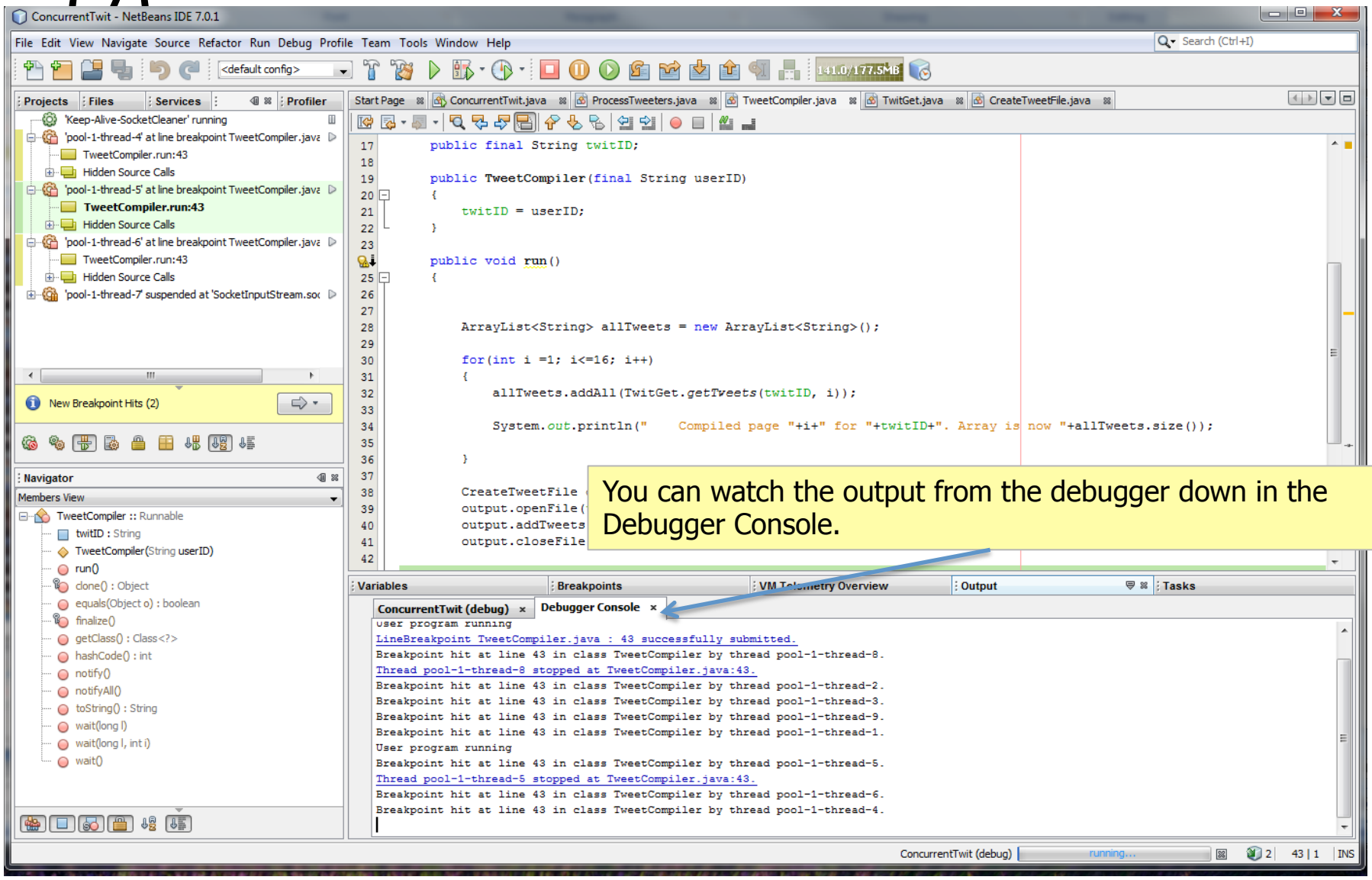
NetBeans – A Brief Tour: Debugging (5)



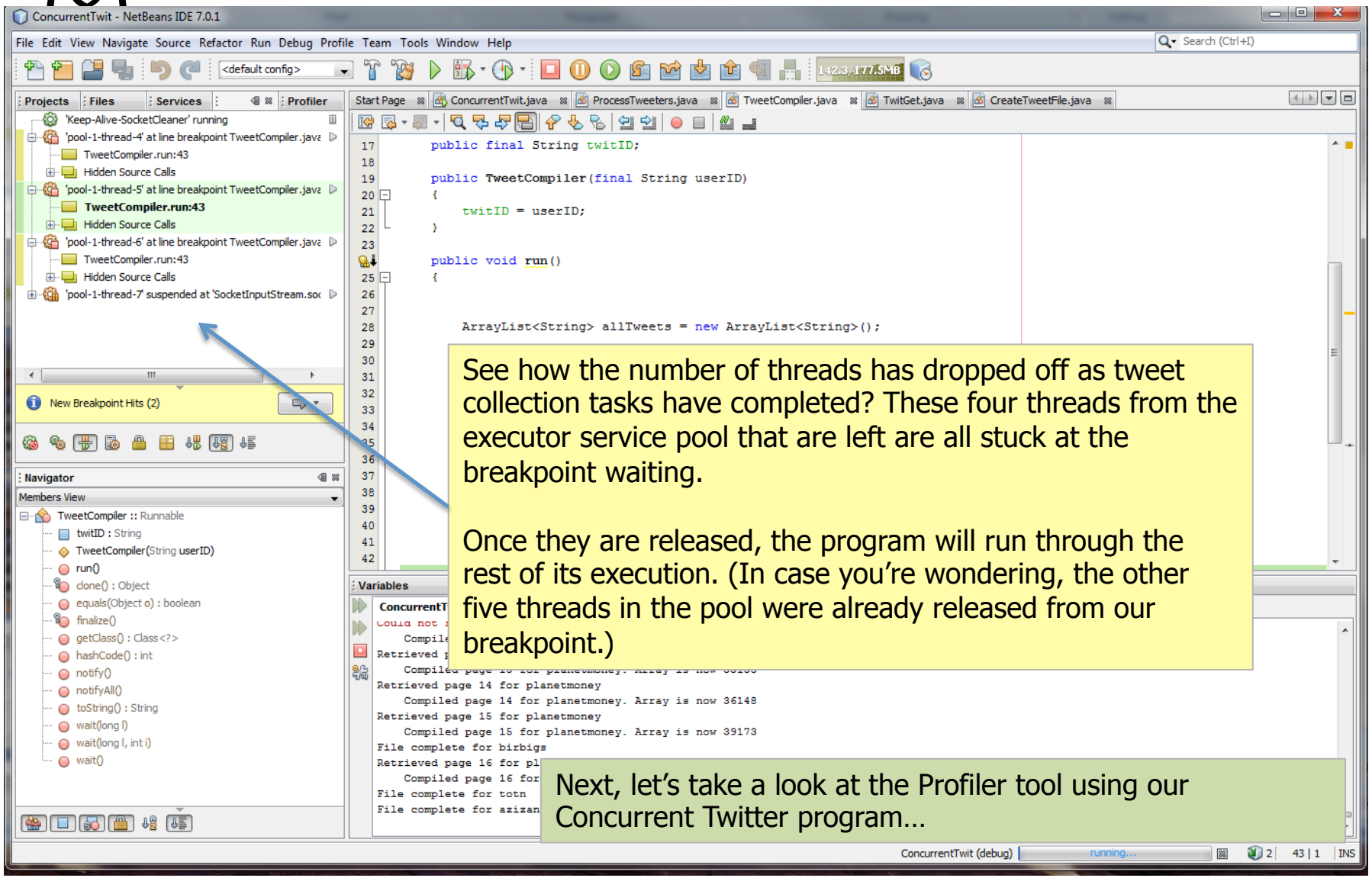
NetBeans – A Brief Tour: Debugging (6)



NetBeans – A Brief Tour: Debugging (7)



NetBeans – A Brief Tour: Debugging (8)

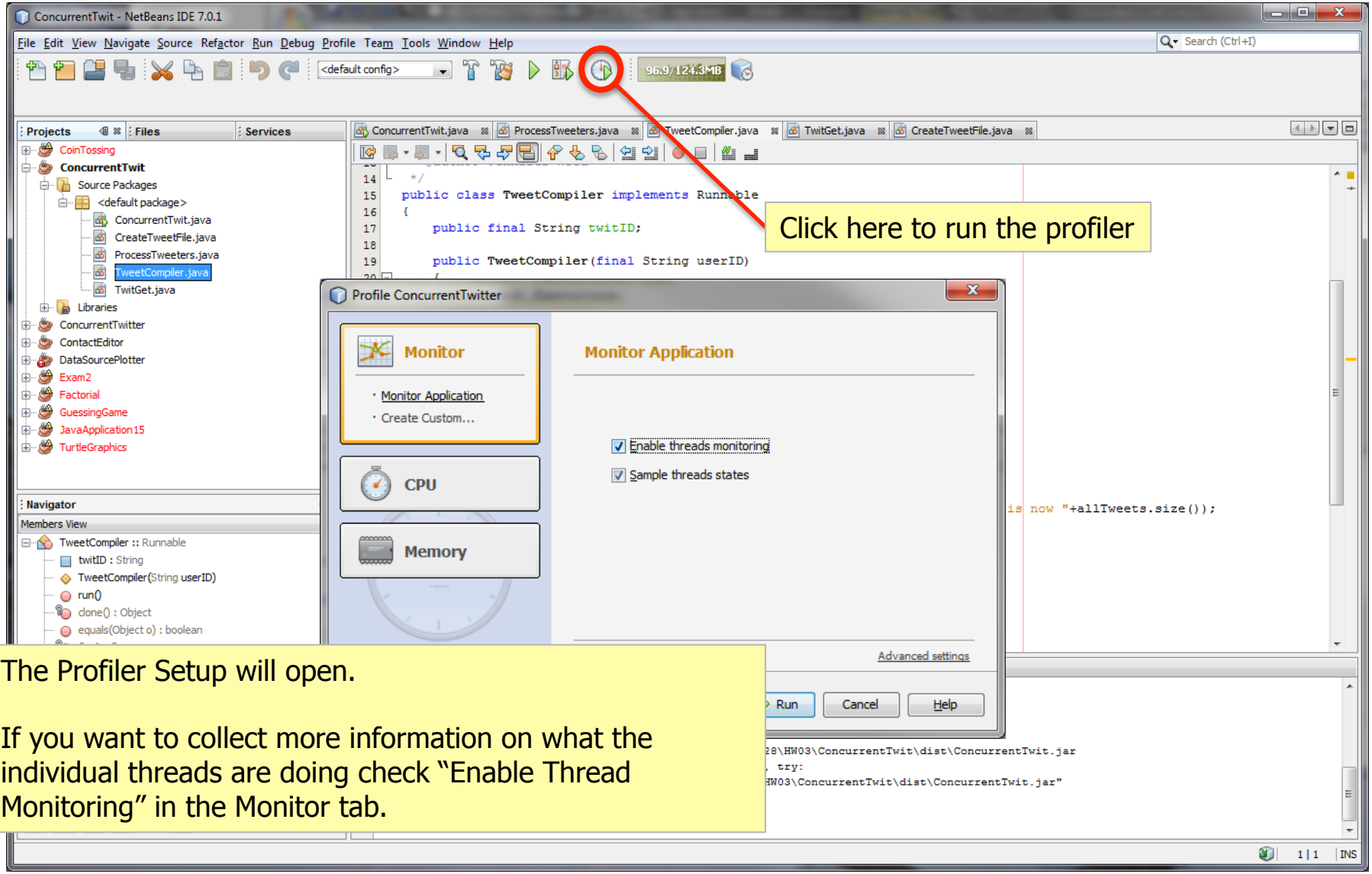


See how the number of threads has dropped off as tweet collection tasks have completed? These four threads from the executor service pool that are left are all stuck at the breakpoint waiting.

Once they are released, the program will run through the rest of its execution. (In case you're wondering, the other five threads in the pool were already released from our breakpoint.)

Next, let's take a look at the Profiler tool using our Concurrent Twitter program…

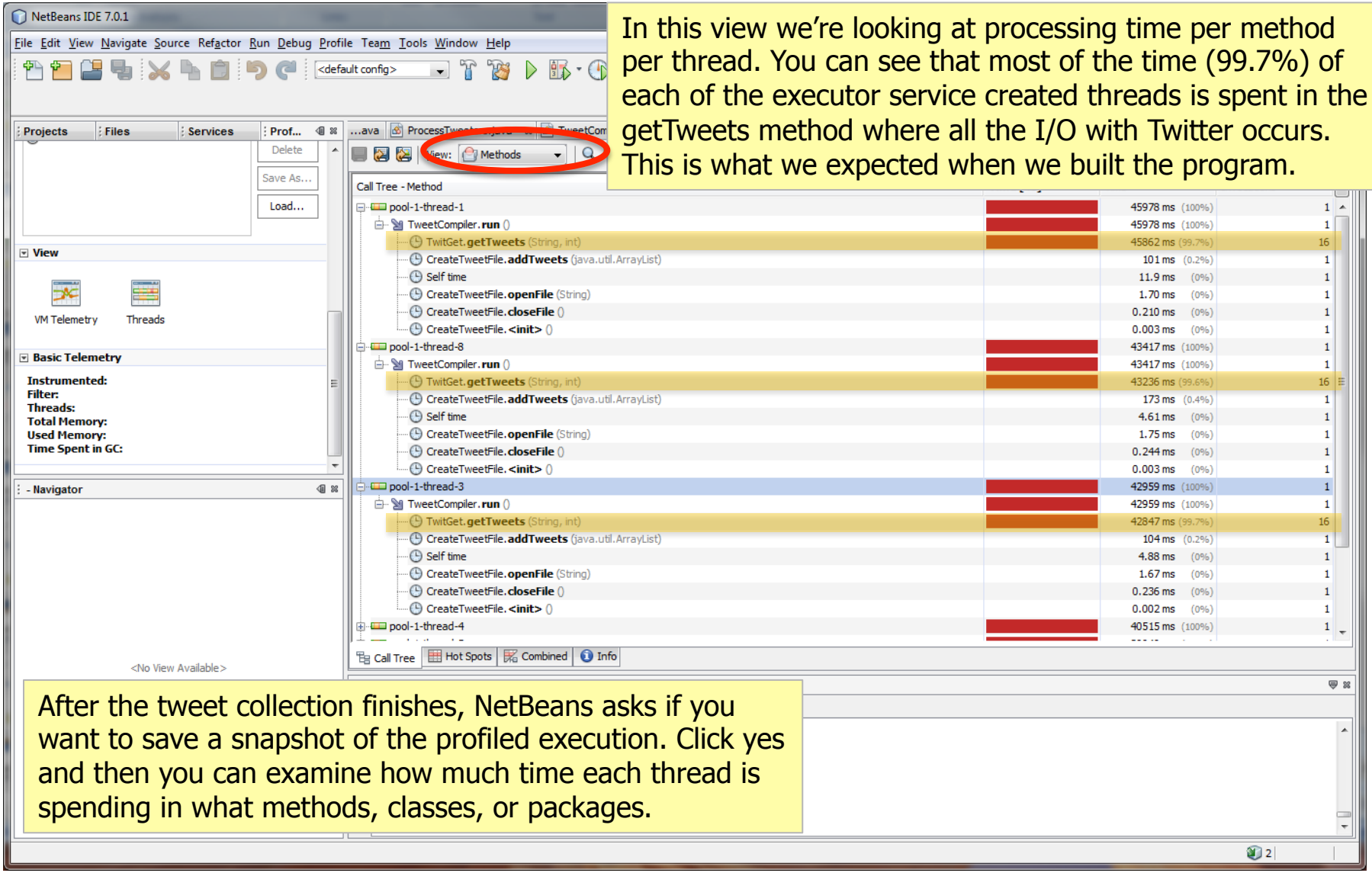
NetBeans – A Brief Tour: Profiler



The Profiler Setup will open.

If you want to collect more information on what the individual threads are doing check "Enable Thread Monitoring" in the Monitor tab.

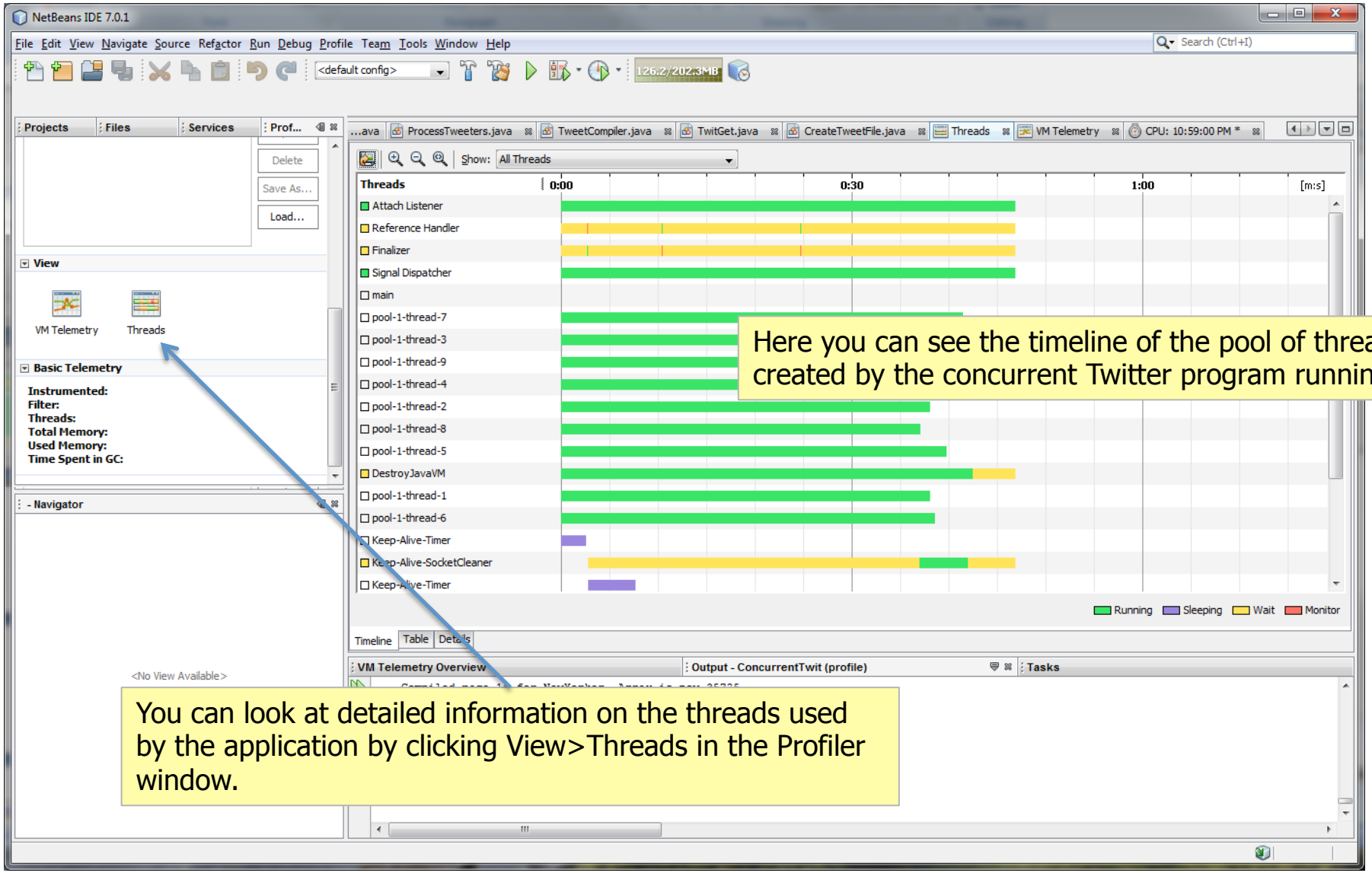
NetBeans – A Brief Tour: Profiler (2)



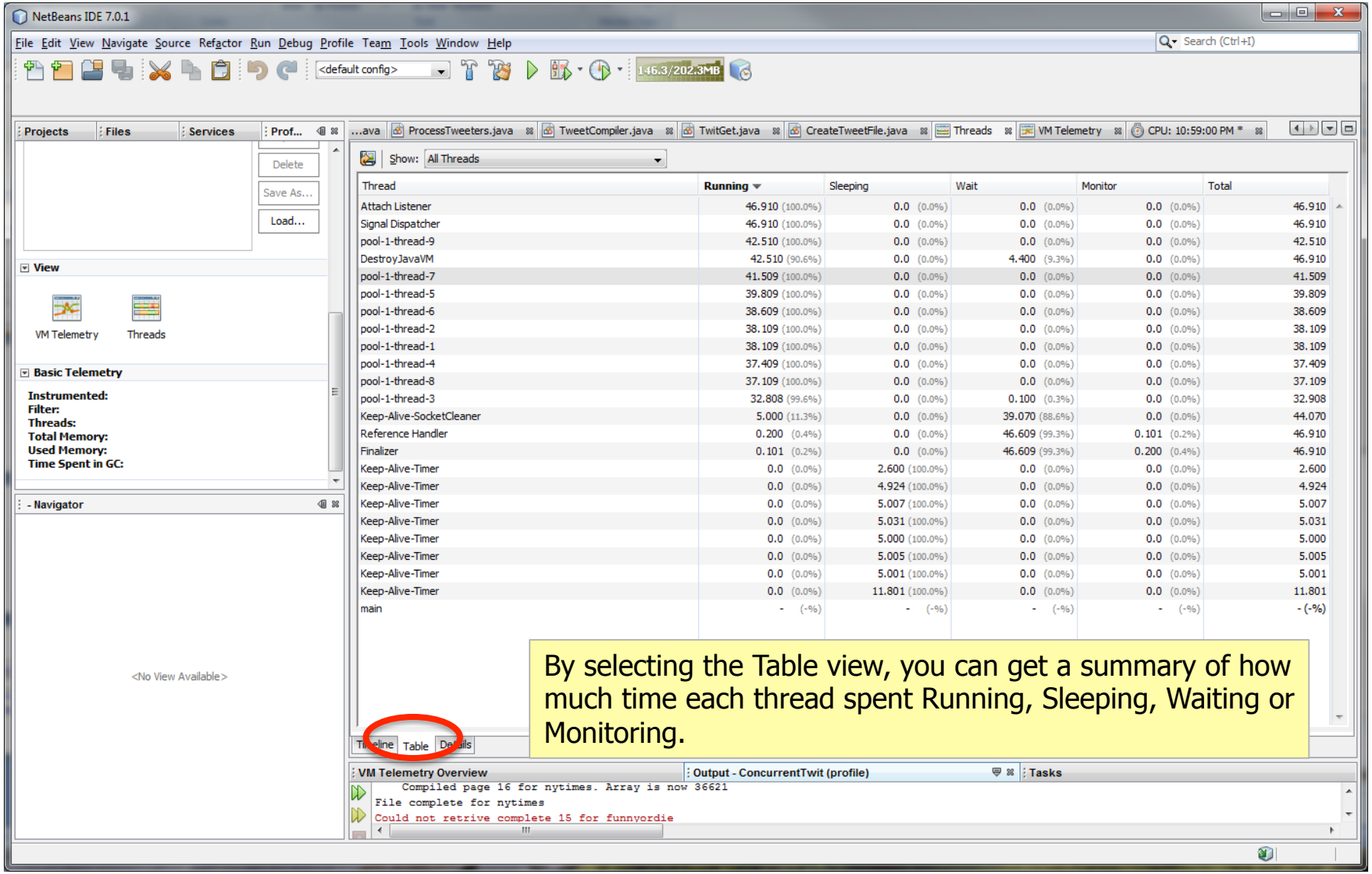
In this view we're looking at processing time per method per thread. You can see that most of the time (99.7%) of each of the executor service created threads is spent in the getTweets method where all the I/O with Twitter occurs. This is what we expected when we built the program.

After the tweet collection finishes, NetBeans asks if you want to save a snapshot of the profiled execution. Click yes and then you can examine how much time each thread is spending in what methods, classes, or packages.

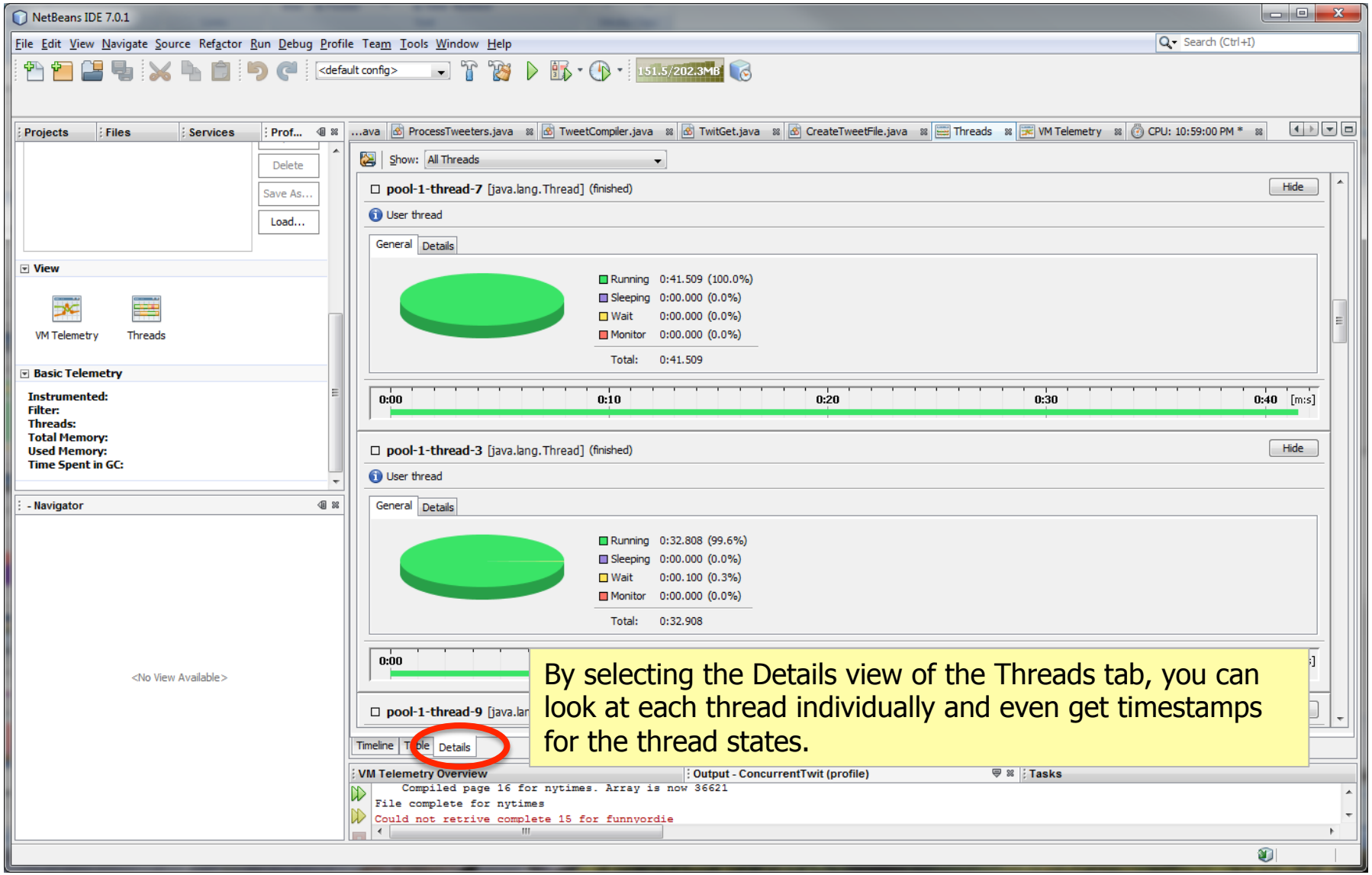
NetBeans – A Brief Tour: Profiler (3)



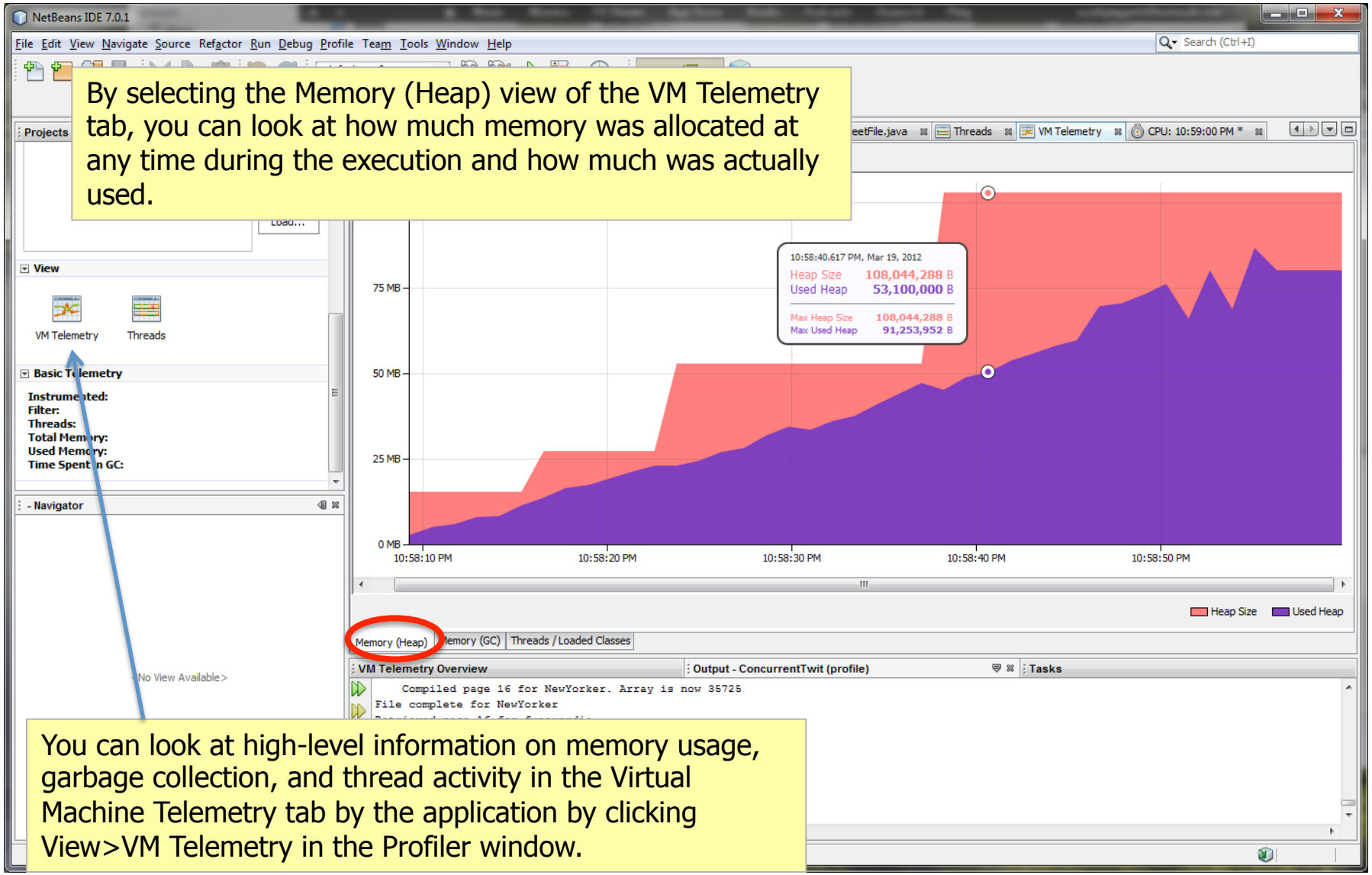
NetBeans – A Brief Tour: Profiler (4)



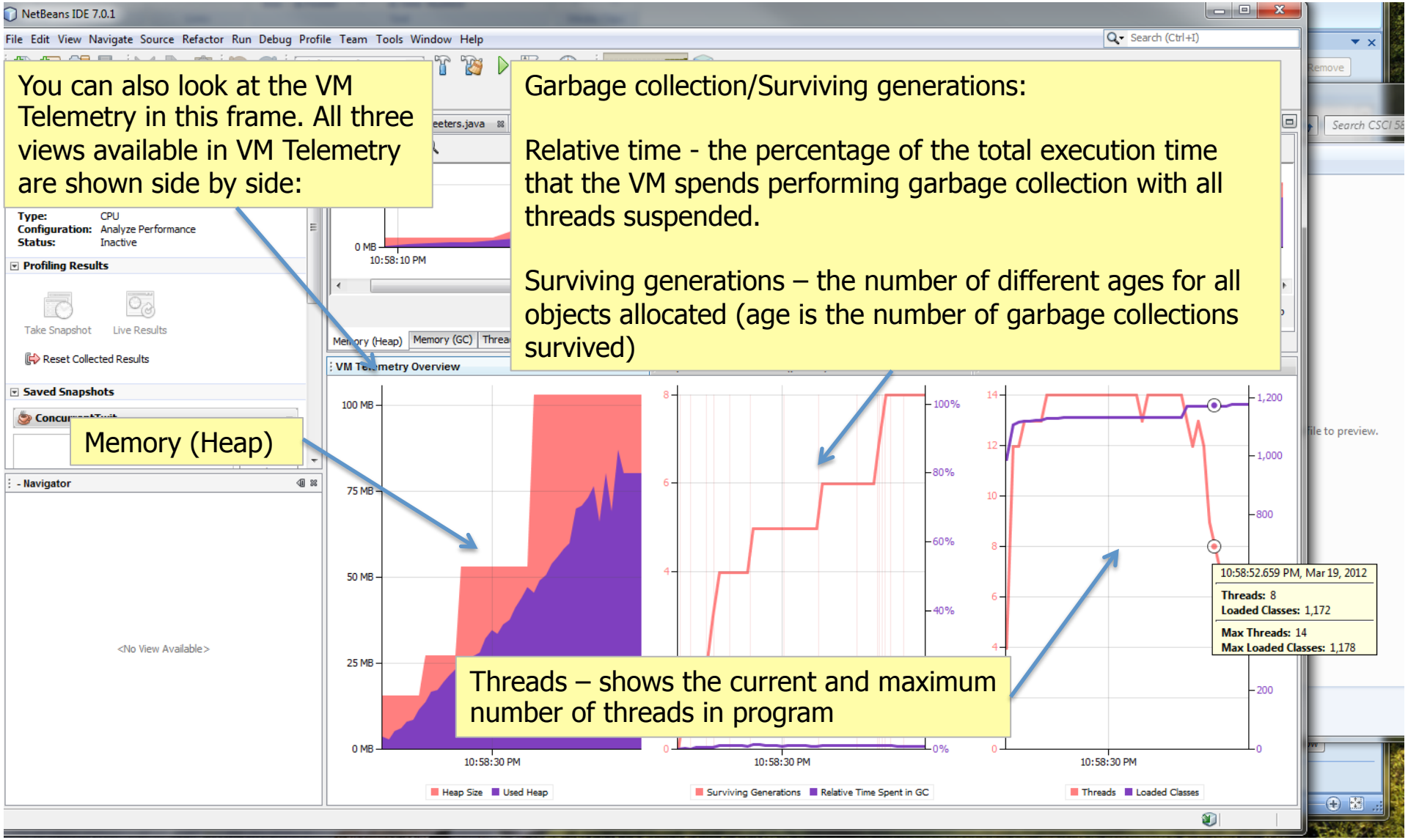
NetBeans – A Brief Tour: Profiler (5)



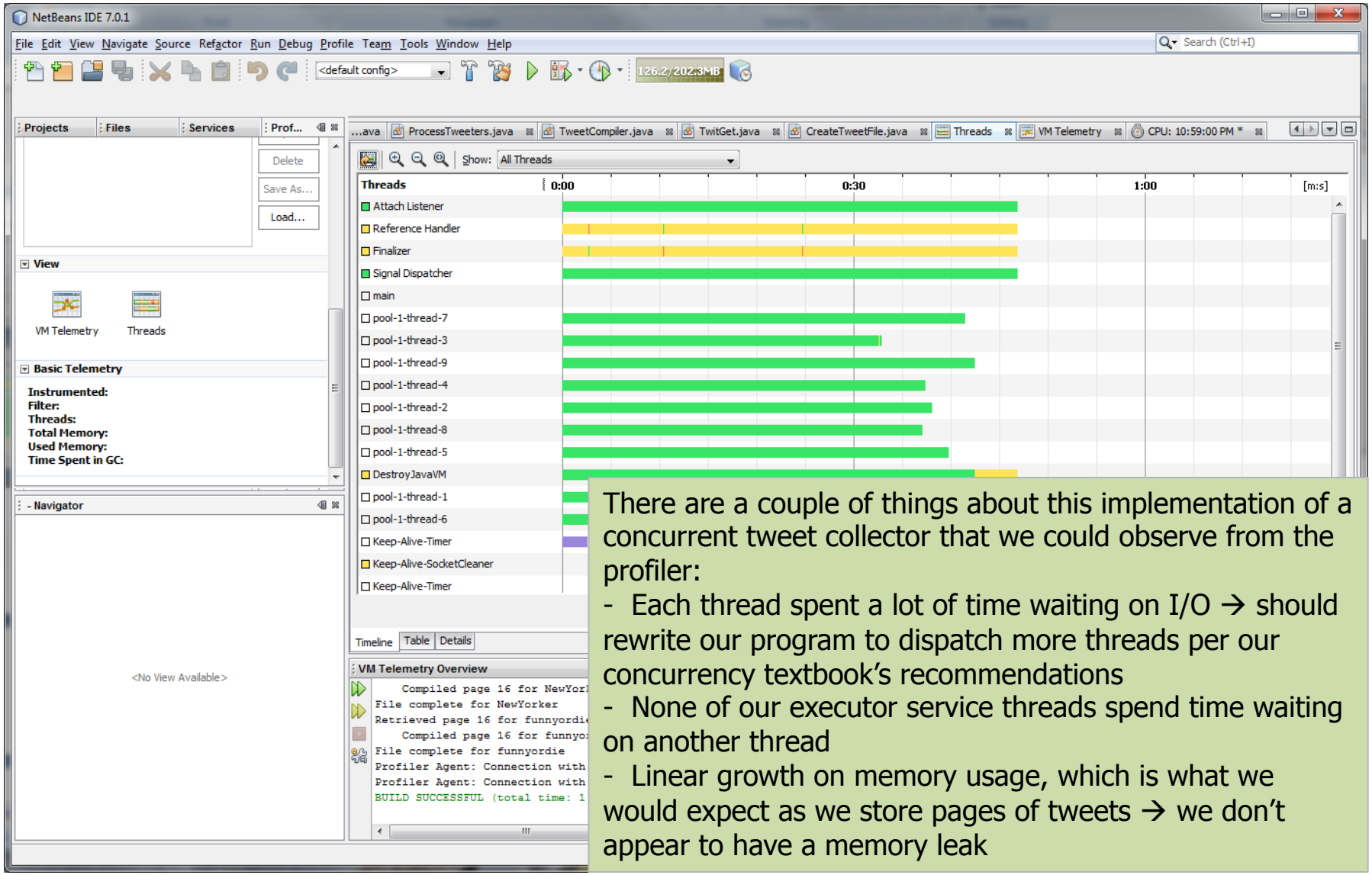
NetBeans – A Brief Tour: Profiler (6)



NetBeans – A Brief Tour: Profiler (7)



NetBeans – A Brief Tour: Profiler (8)



There are a couple of things about this implementation of a concurrent tweet collector that we could observe from the profiler:

- Each thread spent a lot of time waiting on I/O → should rewrite our program to dispatch more threads per our concurrency textbook's recommendations
- None of our executor service threads spend time waiting on another thread
- Linear growth on memory usage, which is what we would expect as we store pages of tweets → we don't appear to have a memory leak

Summary

- Modern Integrated Development Environments provide a high level of functionality for programmers in a unified workspace
- Reduces the need for users to switch views or open other tools while building and testing software
- Use of IDEs has made it easier to prevent accidental difficulties from swamping your development project
- A multiverse of IDEs exists allowing you to find the right one for your programming language that provides the features you need
- NetBeans, primarily used for Java applications, provides much of the support and many of the features that are typical of a modern IDE
  - Learning to use these tools takes time and effort, but can really increase your productivity and the quality of your final product

Sources

- Brooks, Frederick P., "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, Vol. 20, No. 4 (April 1987) pp. 10-19.
- Eclipse Resources. Eclipse.org. Retrieved March 18, 2012 from http://www.eclipse.org/resources/index.php
- Integrated development environment. Wikipedia. Retrieved March 16, 2012 from http://en.wikipedia.org/wiki/Integrated_development_environment
- Intersimone, D. (2010, July 2). The evolution to integrated development environments. *Computerworld.com*. Retrieved March 19, 2012 from http://blogs.computerworld.com/16451/the_evolution_of_the_integrated_development_environment_ide
- NetBeans IDE 7.1 Features. *Netbeans.org.* Retrieved March 18, 2012 from http://netbeans.org/features/index.html
- Nourie, D. (2005, March 24). Getting Started with an Integrated Development Environment. *Sun Developer Network (SDN)*. Retrieved March 16, 2012 from http://java.sun.com/developer/technicalArticles/tools/intro.html
- O'Dell, J. (2010, October 6). A Beginner's Guide to Integrated Development Environments. *Mashable.com*. Retrieved March 17, 2012 from http://mashable.com/2010/10/06/ide-guide/
- Refactoring. *NetBeans Wiki*. Retrieved March 20, 2012 from http://wiki.netbeans.org/Refactoring
- Visual Studio 2010 Comparison. *Microsoft Visual Studio*. Retrieved March 18, 2012 from http://www.microsoft.com/visualstudio/en-us/products/2010-editions/product-comparison
- Walker, Kevin. (2011, November 30). Netbeans Debugger Tutorial. *Ehow.com*. Retrieved March 31, 2012 from http://www.ehow.com/way_6566313_netbeans-debugger-tutorial.html