NetBeans IDE

università di ferrara

DA SEICENTO ANNI GUARDIAMO AVANTI.

Riferimenti

- http://netbeans.org/kb/trails/java-se.html
- http://wiki.netbeans.org/Refactoring

NetBeans IDE

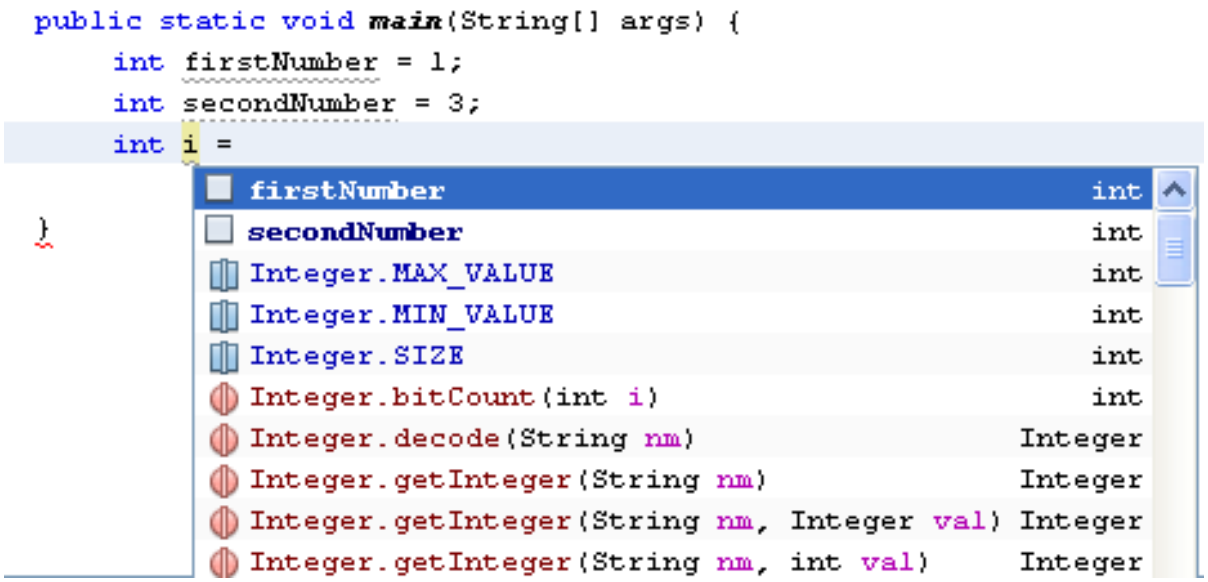
- Live Parsing
- Refactoring
- Smart Code Completion
- Jump to
  - Navigare all'interno del codice
  - Dall'errore alla linea
- Navigare per classi e interfacce

NetBeans IDE

- Swing gui buider
- Profiler
- Debugger
- Version control
- Connessioni a DB
- JavaEE
  - Web Frameworks
  - Web services

Code Completion

- Completamento automatico (ctrl-spazio)



Suggerimento errori

- Netbeans compila e verfica la sintassi del codice “al volo”
- Ci indica subito i possibili errori
- E talvolta anche le probablili soluzioni
  - Es. aggiungere un import, modificare la visibilità

Refactoring

- Refactoring: miglioramento della struttura e dell'organizzazione di un programma senza modificare il suo comportamento
  - Migliora il design del software
  - Rende il codice più comprensibile
  - Facilita la manutenzione
  - Aiuta a trovare gli errori
  - Velocizza la programmazione
- Esempio: "rename" cambia il nome di un metodo o di una variabile in tutte le occorenze

Replace block of code with a method

- Click destro > Refactor > Introduce > Method

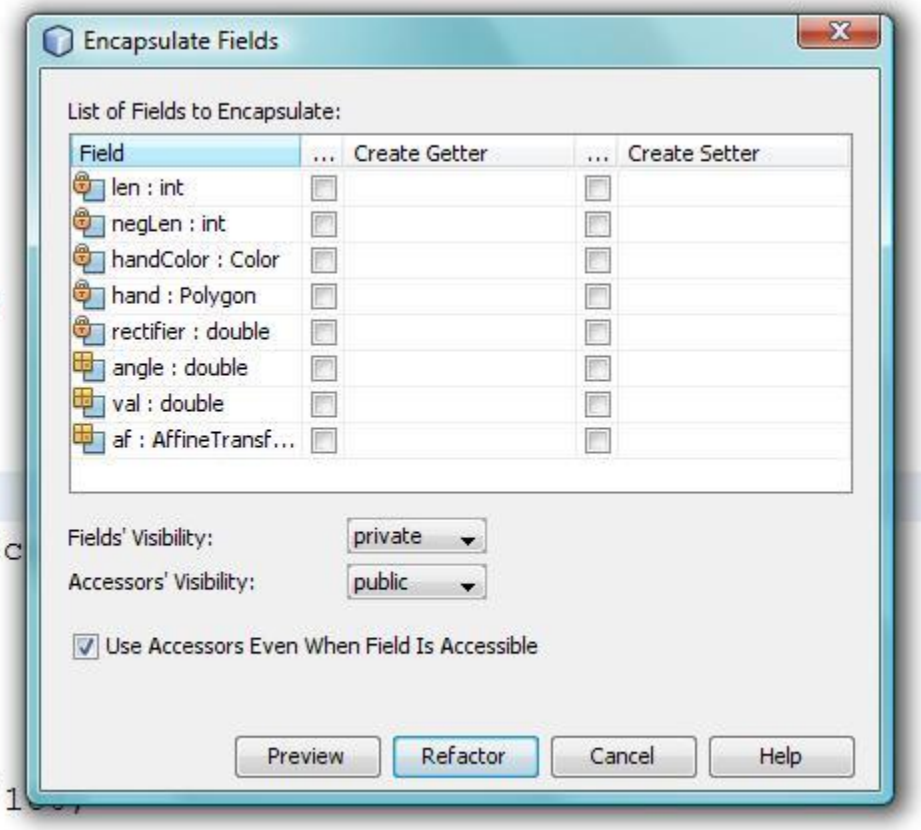
```
public static void main(String[] args) {
    int val=5;
    System.out.print(val);
}
```


```
public static void main(String[] args) {
    newMethod();
}

private static void newMethod() {
    int val = 5;
    System.out.print(val);
}
```

Encapsulate Fields

- Cleck destro > Refactor > Encapsulate Fields

```
class Hand{
        private int len;
        private int negLen;
        private Color handColor;
        private Polygon hand;
        private double rectifier;
        double angle;
        double val;
        AffineTransform af;

Hand(double value,Color col,int rec
        len = 100;
        val = value;
        negLen = 0;
        handColor = col;
        rectifier = Math.PI*rec/1
        initialize();
```



Other frequently used refactoring techniques

- Move Class: Moves a class to another package or into another class. In addition, all source code in your project is updated to reference the class in its new location.
- Safely Delete: Checks for references to a code element and then automatically deletes that element if no other code references it.
- Change Method Parameters: Enables you to add parameters to a method and change the access modifier.
- Extract Interface: Creates a new interface from the selected public non-static methods in a class or interface.
- Extract Superclass: Creates a new abstract class, changes the current class to extend the new class, and moves the selected methods and fields to the new class.
- Move Inner to Outer Level: Moves an inner class or method one level up in hierarchy.

Debugging in NetBeans

Configurable Debugger: In the Options dialog, you can configure breaking/suspending behavior, you can specify Variable Formatters, and skip methods and packages using Step Filters.

Debugging Window: The Debugging window integrates the Sessions, Threads and Call Stack views.

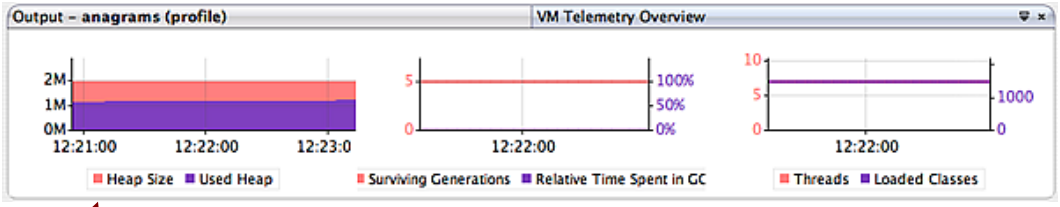
Each sessions is broken down in its **list of threads**, and you can expand each suspended thread to its call stack, etc. You can resume/suspend threads with one click on the play/pause buttons.

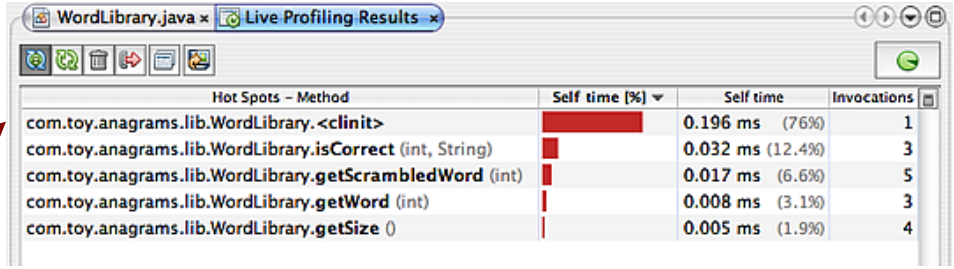
Configurable Breakpoints: In addition to the **standard line and method breakpoints**, the NetBeans debugger provides advanced **Class, Thread, Exception, and Variable breakpoints**. Configure these custom breakpoints to be triggered by conditions and events such as uncaught exceptions, class loading, or variable access.

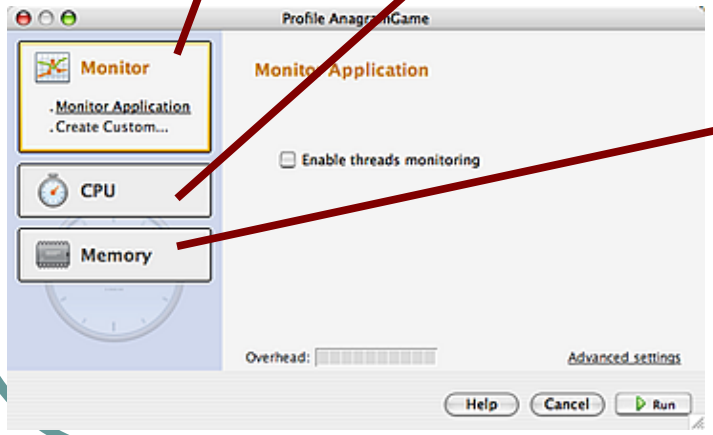
Expression Evaluation: Evaluate Java-syntax expressions assigned to watches and conditional breakpoints "live" while stepping through your code. Moving the pointer over the variable and the current value is evaluated and displayed in a tool tip.

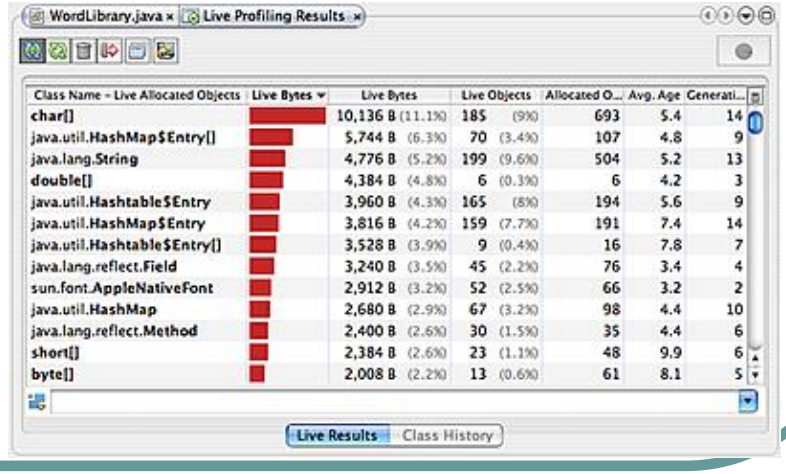
Expression Stepping: You can easily step over individual expressions within a statement. The debugger will display the return value from each expression. The Step Into action (F7) lets you select the method call to step into if there is more than one possibility at the current line.

Profiling


Output – anagrams (profile)
VM Telemetry Overview
2M
1M
0M
12:21:00
12:22:00
12:23:0
5
0
100%
50%
0%
12:22:00
10
5
0
1000
0
12:22:00
Heap Size
Used Heap
Surviving Generations
Relative Time Spent in GC
Threads
Loaded Classes


WordLibrary.java
Live Profiling Results
Hot Spots – Method
Self time [%]
Self time
Invocations
com.toy.anagrams.lib.WordLibrary.<clinit> 0.196 ms (76%) 1
com.toy.anagrams.lib.WordLibrary.isCorrect (int, String) 0.032 ms (12.4%) 3
com.toy.anagrams.lib.WordLibrary.getScrambledWord (int) 0.017 ms (6.6%) 5
com.toy.anagrams.lib.WordLibrary.getWord (int) 0.008 ms (3.1%) 3
com.toy.anagrams.lib.WordLibrary.getSize () 0.005 ms (1.9%) 4


Profile AnagramGame
Monitor
Monitor Application
Create Custom...
CPU
Memory
Monitor Application
Enable threads monitoring
Overhead:
Advanced settings
Help
Cancel
Run


WordLibrary.java
Live Profiling Results
Class Name – Live Allocated Objects
Live Bytes
Live Bytes
Live Objects
Allocated O...
Avg. Age
Generati...
char[] 10,136 B (11.1%) 185 (9%) 693 5.4 14
java.util.HashMap$Entry[] 5,744 B (6.3%) 70 (3.4%) 107 4.8 9
java.lang.String 4,776 B (5.2%) 199 (9.6%) 504 5.2 13
double[] 4,384 B (4.8%) 6 (0.3%) 6 4.2 3
java.util.Hashtable$Entry 3,960 B (4.3%) 165 (8%) 194 5.6 9
java.util.HashMap$Entry 3,816 B (4.2%) 159 (7.7%) 191 7.4 14
java.util.Hashtable$Entry[] 3,528 B (3.9%) 9 (0.4%) 16 7.8 7
java.lang.reflect.Field 3,240 B (3.5%) 45 (2.2%) 76 3.4 4
sun.font.AppleNativeFont 2,912 B (3.2%) 52 (2.5%) 66 3.2 2
java.util.HashMap 2,680 B (2.9%) 67 (3.2%) 98 4.4 10
java.lang.reflect.Method 2,400 B (2.6%) 30 (1.5%) 35 4.4 6
short[] 2,384 B (2.6%) 23 (1.1%) 48 9.9 6
byte[] 2,008 B (2.2%) 13 (0.6%) 61 8.1 5
Live Results
Class History