

Programowanie w Javie

Andrzej Czajkowski

Lista nr 0 – Debugger w Javie

Celem ćwiczenia jest poznanie podstawowych funkcji narzędzia debugera (odpluskwiacz) w środowisku Eclipse. Po ukończeniu ćwiczenia student powinien umieć samodzielnie obsługiwać narzędzie i wykonywać podstawowe instrukcje służące prześledzeniu wykonywania kolejnych etapów programu jak i warunkowego zatrzymania wykonywania programu oraz śledzenia zmian wartości obserwowanych zmiennych. Dzięki czemu student powinien umieć odnajdywać błędy zarówno skutkujące przerwaniem programu jak i błędy skutkujące niepoprawnym wynikiem zwróconym przez program. Laboratorium powstało w oparciu o instrukcję na stronie <http://www.vogella.de/articles/EclipseDebugging/article.html> .

1. Przygotowanie do ćwiczenia.

1.1. Przed ćwiczeniem należy sprawdzić ustawienia interaktywnej pomocy pakietu Eclipse, to jest konfigurację dokumentacji dla biblioteki rt.jar (**Properties**). Jeśli będzie to konieczne, to ustawić ścieżki do pliku ze źródłami JRE oraz dokumentacją API (Javadoc). Przykładowe ustawienia dla źródła (source):

D:/java/src.zip

ustawienie dla Javadoc z Internetu:

<http://download.oracle.com/javase/6/docs/api/>

Deklaracje danego typu lub funkcji można podglądać za pomocą klawiszy Shift+F2 (**Navigate/Open External Javadoc**), a źródło przyciskając prawym klawiszem myszy i wybierając opcję **Open Declaration**.

1.2. Sprawdzić ustawienia zmiennych środowiskowych (CLASSPATH).

1.3. Ustawić ścieżkę dla generatora dokumentacji Javadoc. W tym celu w oknie **Window/Preferences**, w podkatalogu Javadoc należy wpisać ścieżkę do generatora Javadoc np. D:\java\bin\javadoc.exe.

2. Przebieg Ćwiczeń.

Utworzyć nowy projekt Java (File->New-> Project || Java Project) o nazwie "de.vogella.debug.first".

Utworzyć nowy pakiet (package) "de.vogella.debug.first" w projekcie. Następnie utworzyć w pakiecie klasy Counter, Main o poniższej zawartości:

```
package de.vogella.debug.first;

public class Counter {
    private int result=0;
    public int getResult() {
        return result;
    }

    public void count() {
        for (int i = 0; i < 100; i++) {
            result += i +1;
        }
    }
}
```

```

package de.vogella.debug.first;

public class Main {

    /**
     * @param args
     */

    public static void main(String[] args) {
        Counter counter = new Counter();
        counter.count();
        System.out.println("We have counted " + counter.getResult());
    }

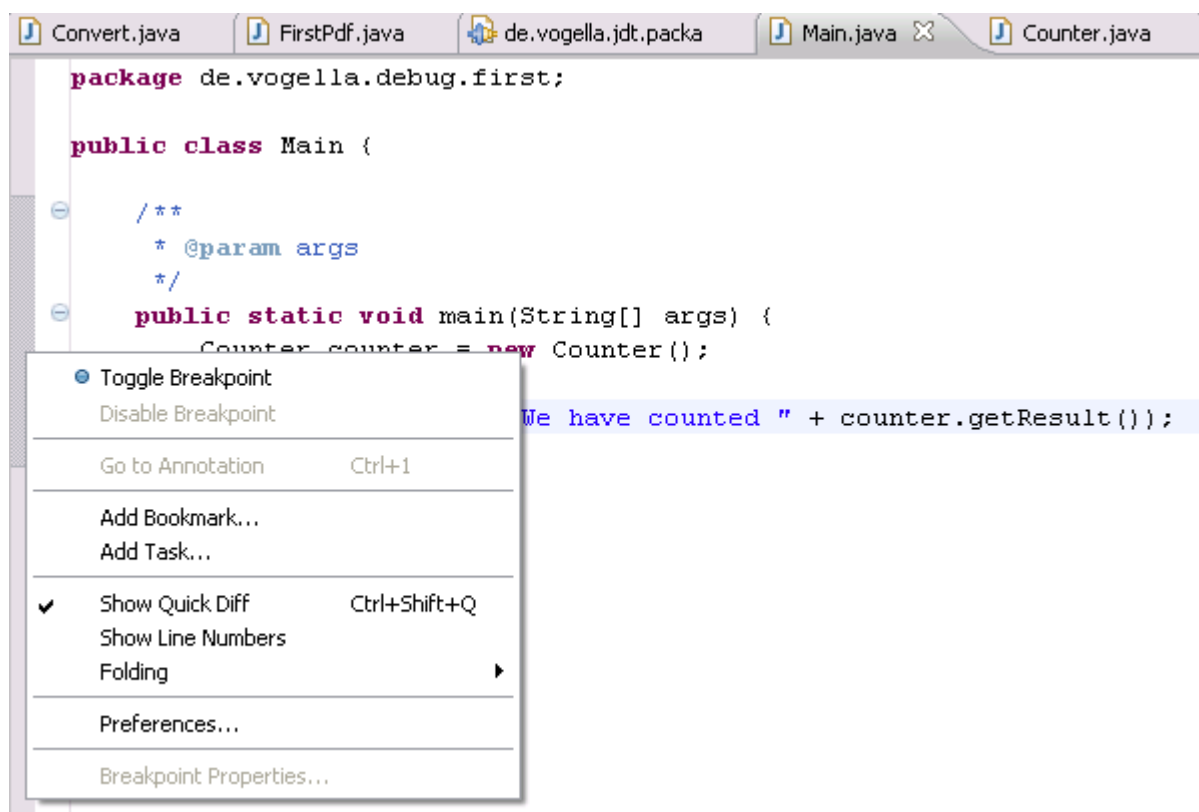
}

```

3. Debugowanie

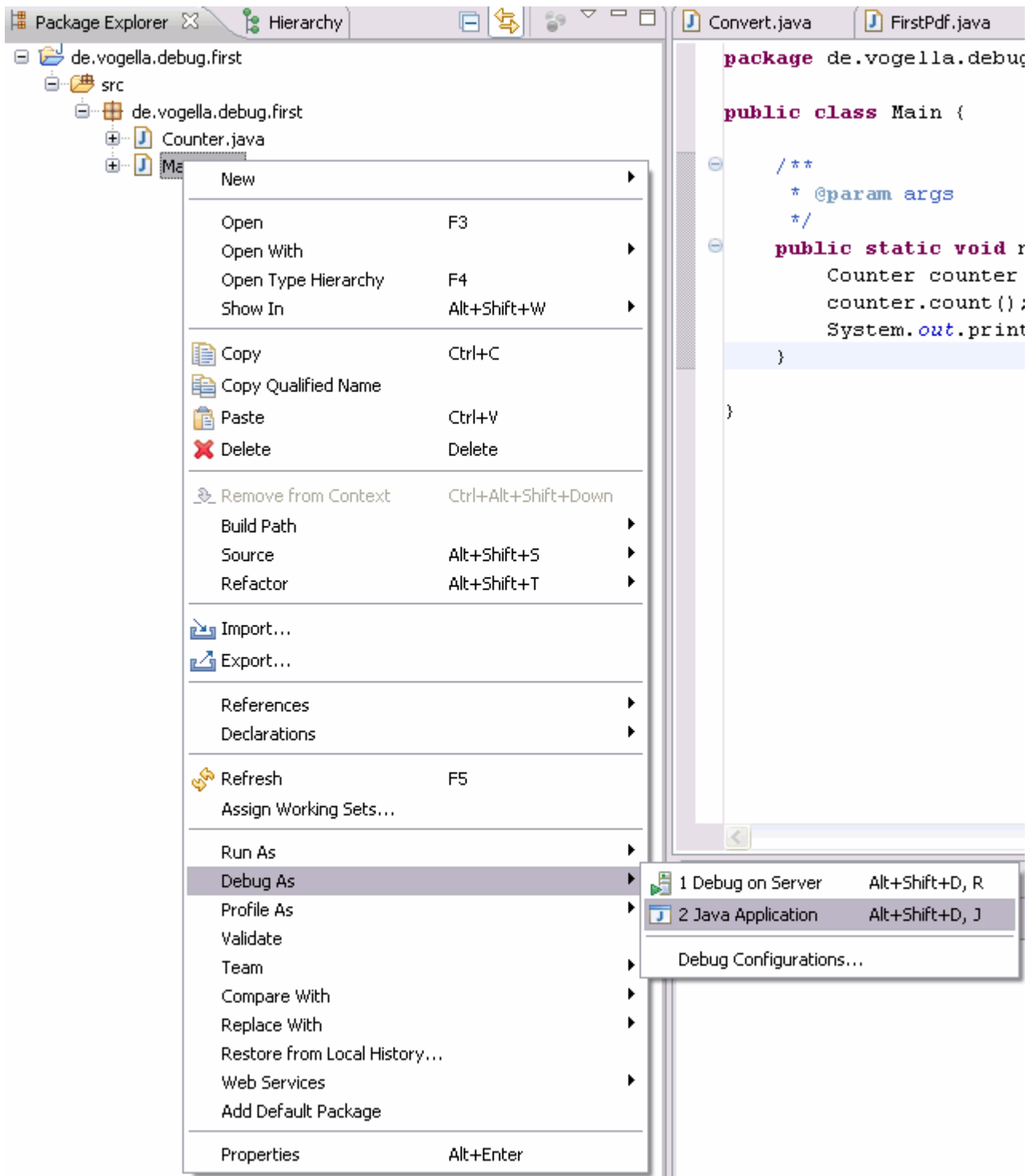
3.1. Ustawianie punktów stopu (Breakpoints).

Aby ustawić punkt wstrzymania wykonywania należy kliknąć prawym przyciskiem myszy w wąskim pasku po lewej stronie edytora kodu i wybrać opcję toggle breakpoint, lub dwukrotnie kliknąć w żądanym miejscu.



3.2. Uruchamianie Odpluskwiacza

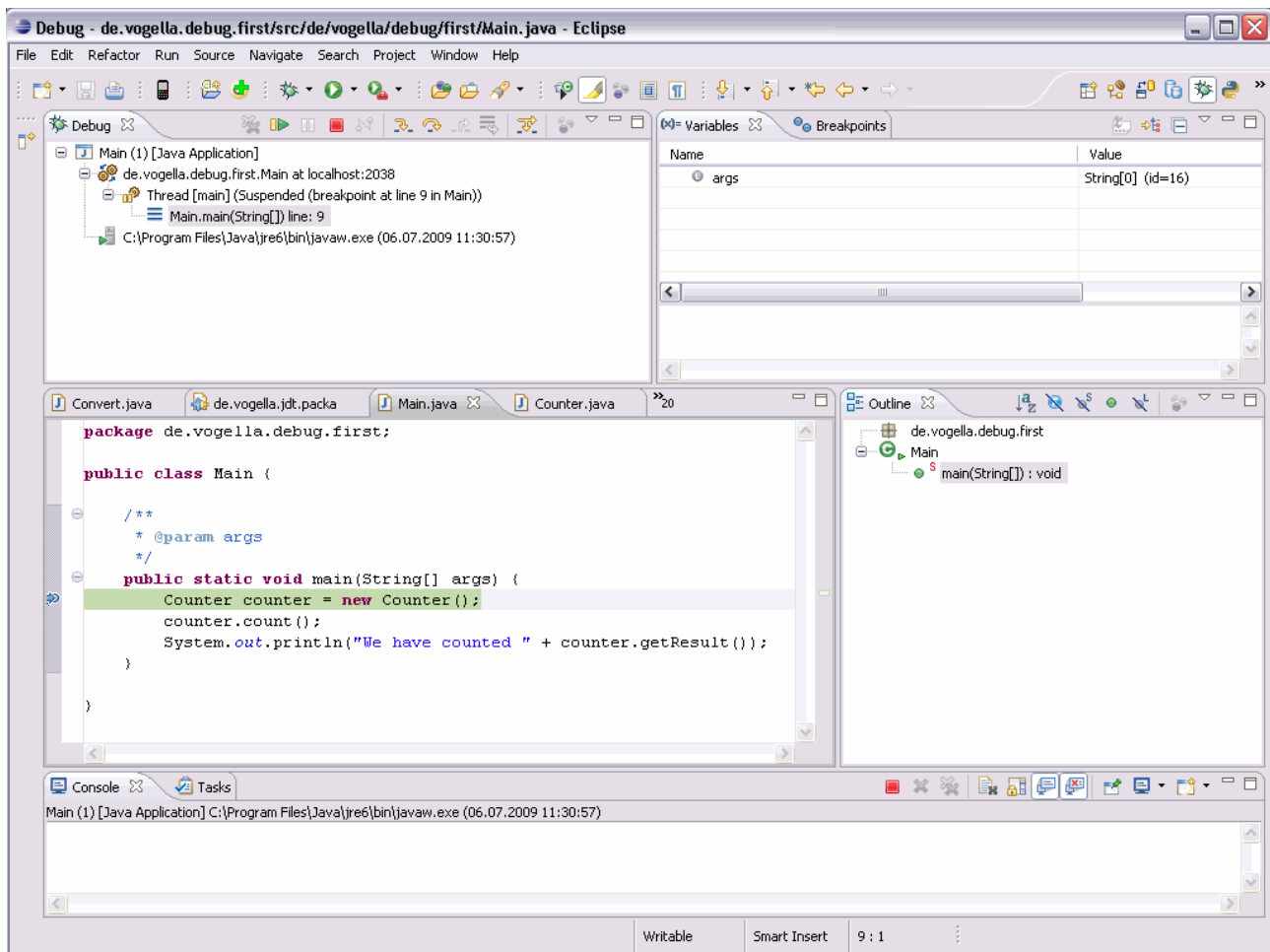
Aby debugować aplikację należy wybrać plik, który zawiera metodę main poprzez kliknięcie prawym przyciskiem myszy i wybranie opcji Debug As -> Java Application.



Wskazówka

Jeśli nie zostały ustawione żadne punkty stopu program wykona się normalnie. Aby debugować program należy zdefiniować punkty wstrzymania wykonania (breakpoints).

Przy pierwszym uruchomieniu debugera, środowisko Eclipse zapyta się czy zmienić perspektywę (widok – układ okien) środowiska na tryb debugowania (aby przełączać się pomiędzy perspektywami należy skorzystać z przycisków w prawym górnym rogu środowiska). Po zatwierdzeniu zmiany widoku układ okien środowiska powinien wyglądać jak na poniższym zdjęciu:

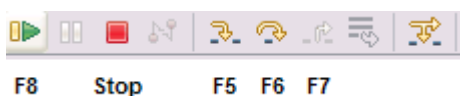


Można używać skrótów F5 / F6, F7 i F8 aby prześledzić wykonywanie kodu.

Table 1. Skróty klawiszowe debugowania

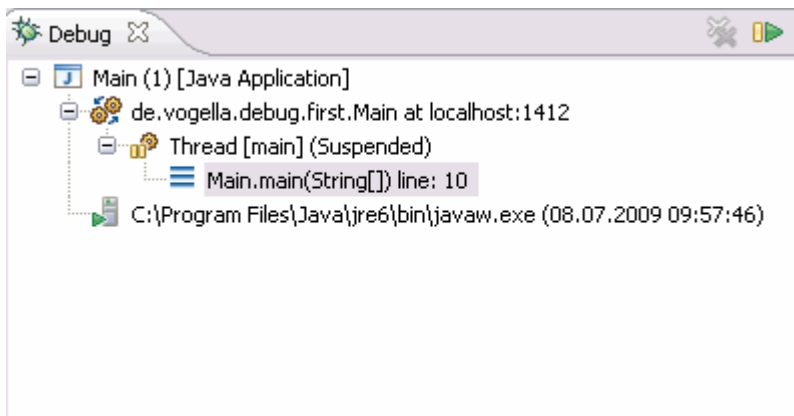
skrót	opis
F5	Kolejny krok wykonywania programu. Jeśli kolejnym krokiem w programie jest metoda/funkcja skrót wywoła instrukcję wewnątrz kodu danej metody (należy pamiętać o dodaniu źródeł bibliotek wbudowanych w celu śledzenia ich wykonywania).
F6	F6 omija śledzenie wykonywania kodu (step over) wywoływanej w danym kroku metody/funkcji.
F7	F7 wykonuje powrót do funkcji która wywołała śledzoną w danej chwili metodę. Spowoduje to zaprzestanie śledzenia wykonywania aktualnie metody.
F8	Skrót F8 powoduje przejście do kolejnego punktu stopu. W przypadku braku kolejnego punktu stopu spowoduje do normalne (nie debugowane) wykonanie aplikacji aż do jej zakończenia.

Odpowiedniki przedstawionych skrótów można znaleźć na interfejsie środowiska:



3.3. Stos

Aktualny stos wykonywanego programu jest wyświetlony w zakładce Debug.

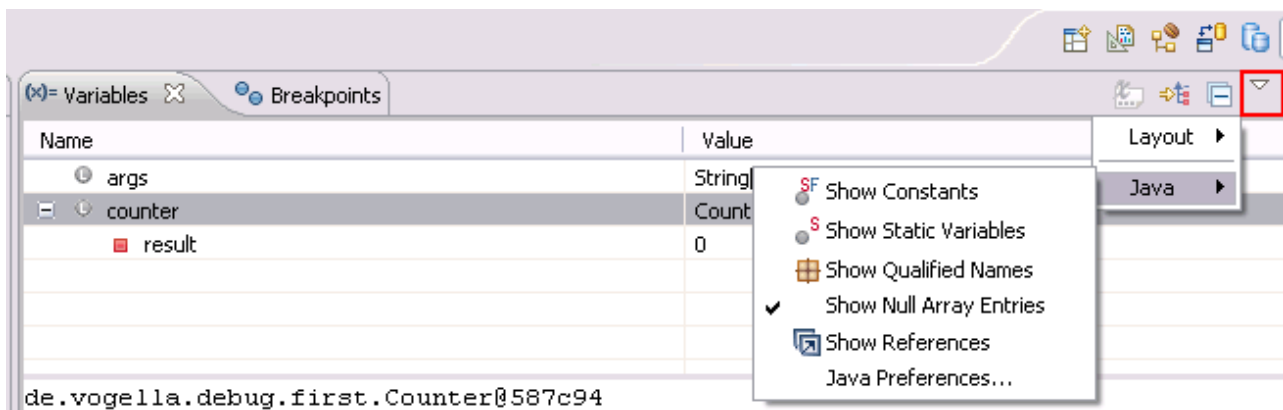


3.4. Zmienne

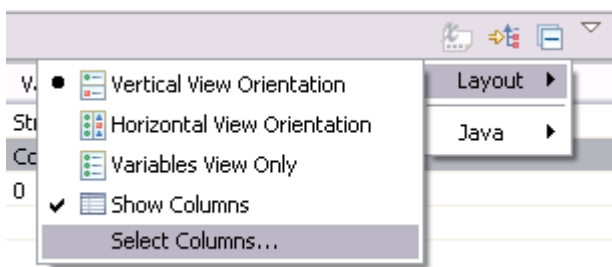
Widok "Variables" wyświetla pola oraz zmienne lokalne dla obecnie podglądanego stosu.

Name	Value
args	String[0] (id=16)
counter	Counter (id=20)
result	0

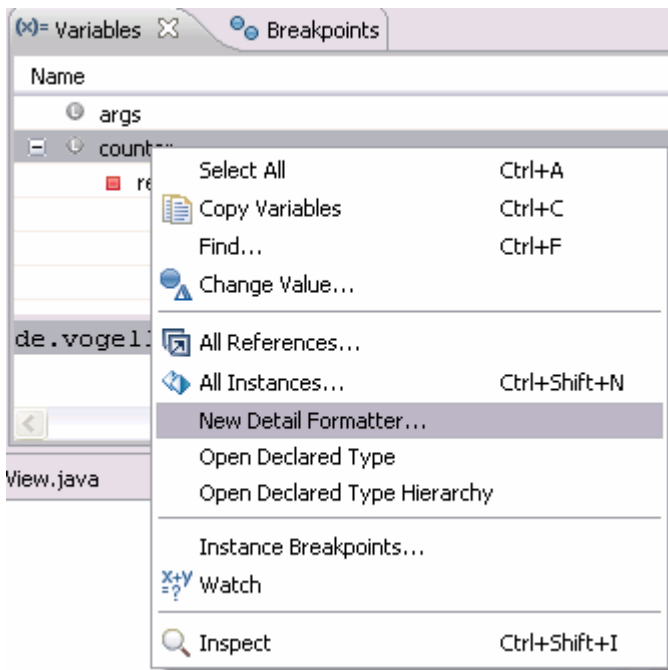
Aby wyświetlić zmienne statyczne należy użyć menu podręcznego:



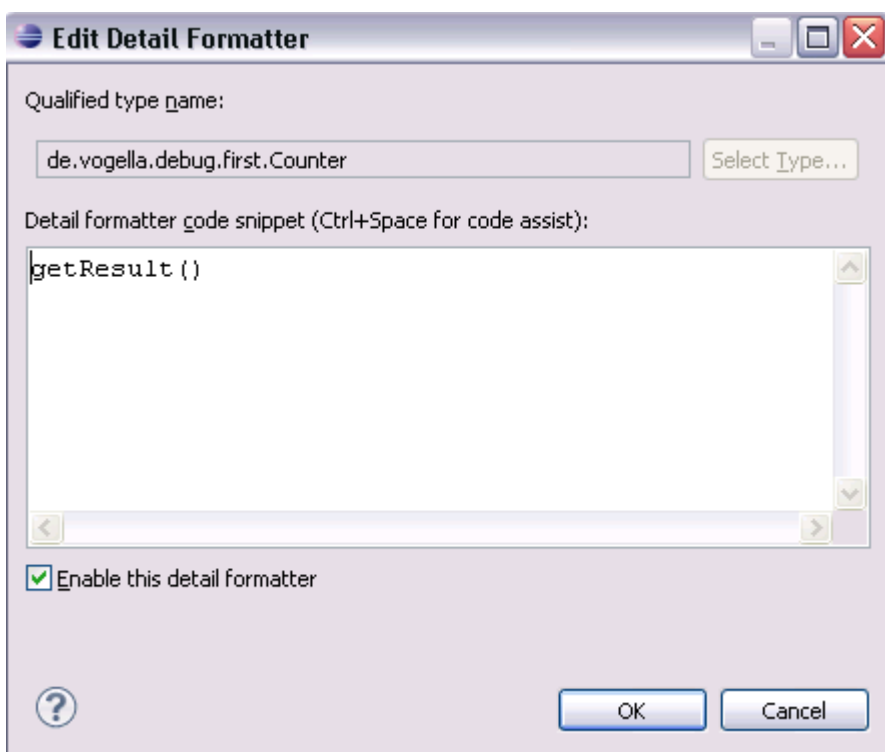
Poprzez menu można spersonalizować wyświetlane kolumny:



Bardzo przydatną funkcją jest opcja "New Detail Formater" dzięki której można zdefiniować w jaki sposób zmienne są wyświetlane. Dla przykładu obiekt counter jest opisany nic nie mówiącym adresem w pamięci np. "de.vogella.debug.first.Counter@587c94" (wynik działania domyślnego wywołania metody toString()). Możemy zmienić wartość wyświetlaną poprzez menu podręczne i opcję "New Details Formater".



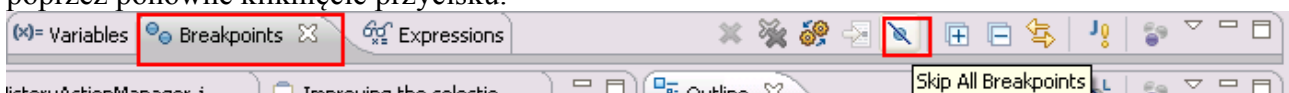
Poprzez sprecyzowanie opisu dla danego obiektu możemy ustawić aby wyświetlała się w trybie debuggera interesująca nas wartość odpowiadająca obiektowi.



4. Zaawansowane debugowanie.

4.1. Pominięcie wszystkich punktów wstrzymania.

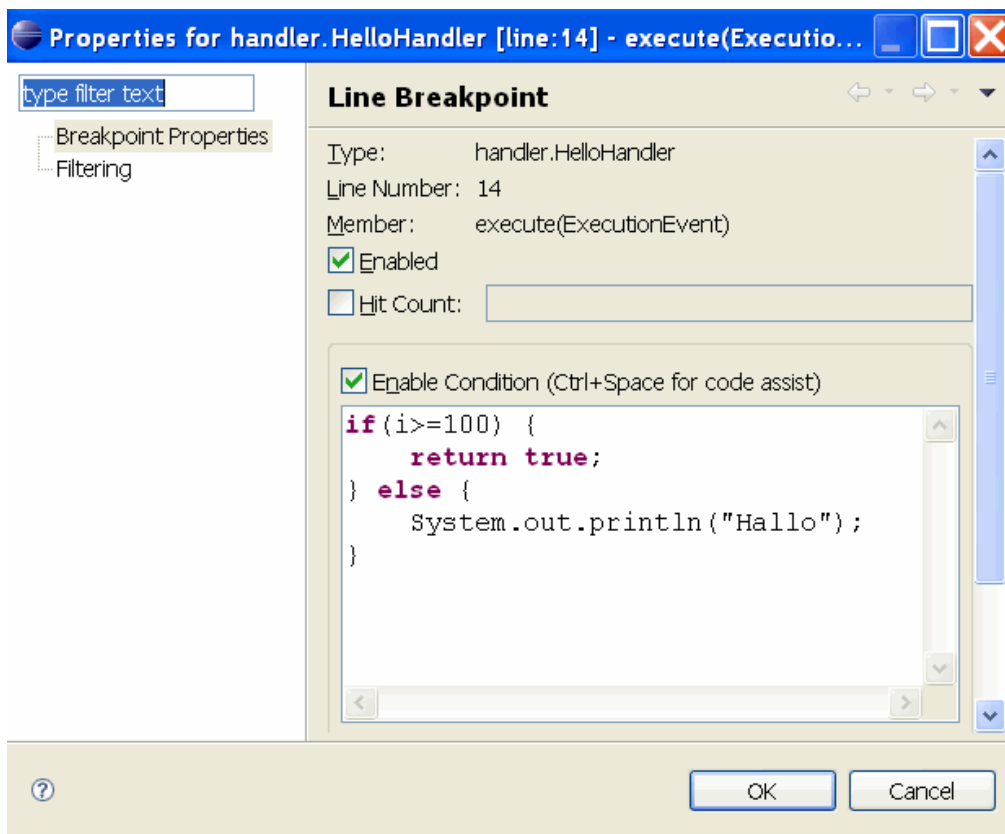
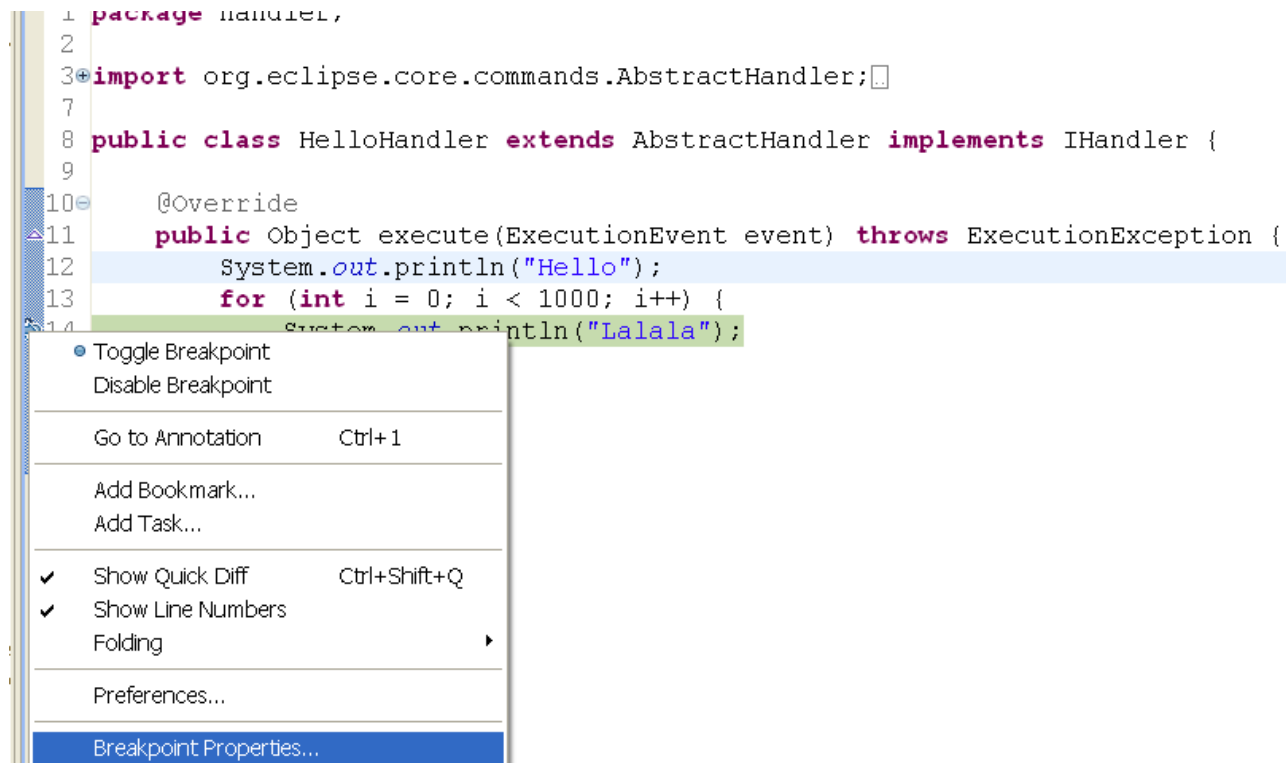
Aby tymczasowo dezaktywować wszystkie punkty wstrzymania, można użyć opcji "Skip all breakpoints", która znajduje się na zakładce punktów zatrzymania. Anulować opcję tą można poprzez ponowne kliknięcie przycisku.



4.2. Właściwości punktów wstrzymania.

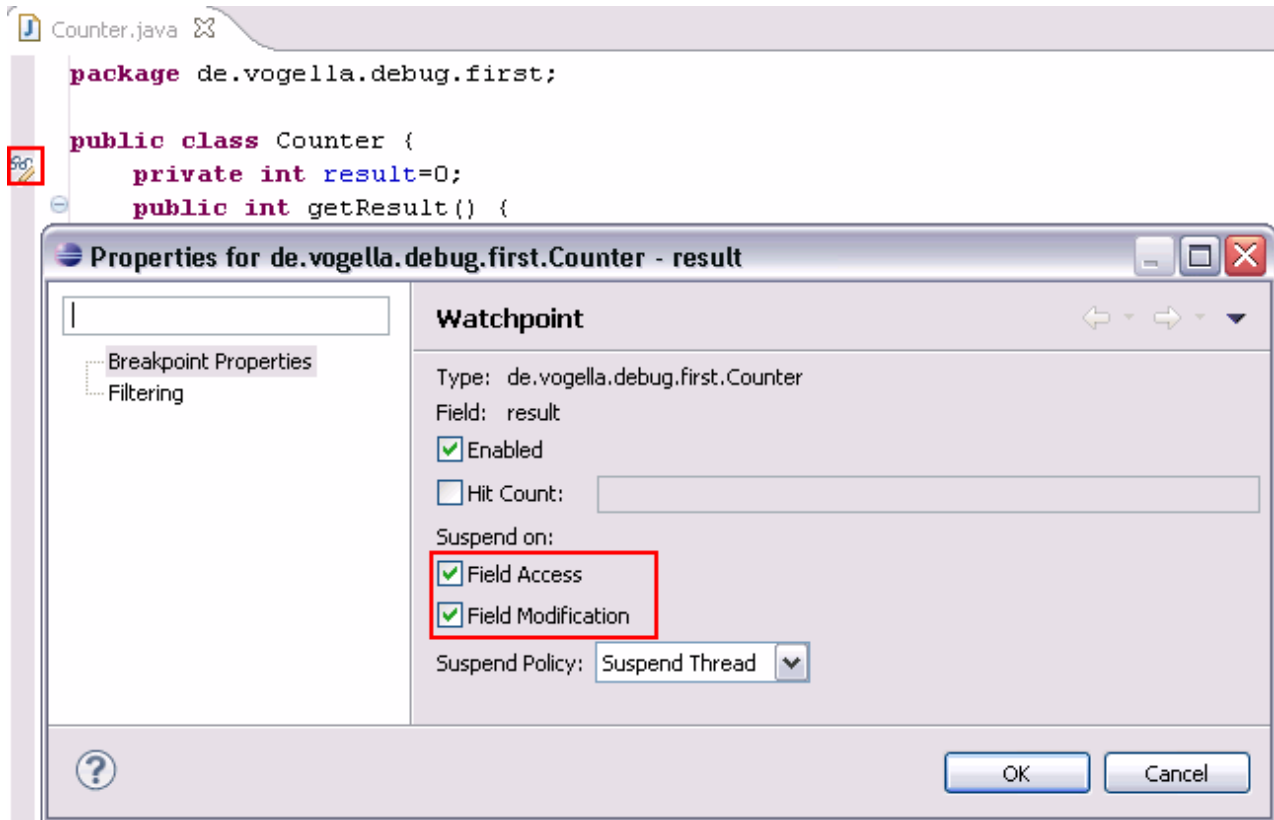
Po ustawieniu punktu wstrzymania w kodzie można zmienić jego właściwości. Dla przykładu można ustawić kiedy punkt wstrzymania ma być uwzględniany. W ustawieniach można na przykład ograniczyć wstrzymywanie wykonania w danym miejscu gdy dana linijka kodu zostanie wykonana 12 razy (Hit Count). Dodatkowo można wprowadzić instrukcje warunkowe ułatwiające debugowanie.

```
1 package handler,  
2  
3 import org.eclipse.core.commands.AbstractHandler;□  
7  
8 public class HelloHandler extends AbstractHandler implements IHandler {  
9  
10 @Override  
11 public Object execute(ExecutionEvent event) throws ExecutionException {  
12     System.out.println("Hello");  
13     for (int i = 0; i < 1000; i++) {  
14         System.out.println("Lalala");  
15     }  
16 }  
17 }
```



4.3. Punkt obserwacji (Watchpoint)

Punktem obserwacji jest to rodzaj punktu wstrzymania dzięki któremu można śledzić zmiany lub odczyty zmiennych lub pól klas. Aby ustawić punkt obserwacji poprzez podwójne kliknięcie na polu po lewej stronie od deklaracji pola. Poprzez menu właściwości można zdefiniować czy wykonanie powinno być wstrzymywane przy odczycie (Field Access) czy modyfikacji pola (Field Modification).



4.4. Punkt wstrzymania przy wystąpieniu wyjątku (Exception Breakpoint)

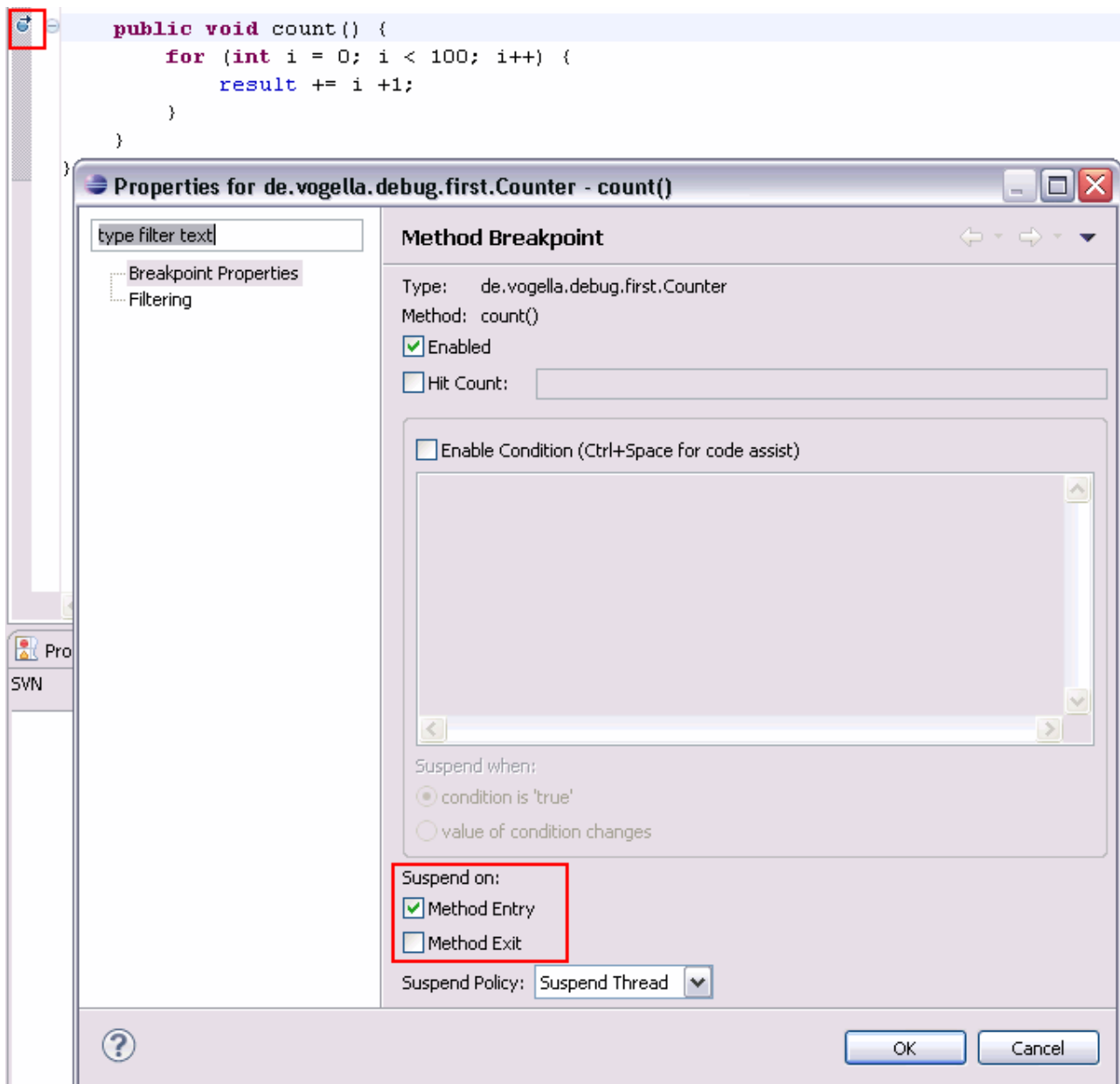
Zastosowaniem tego rodzaju wstrzymania jest wstrzymanie wykonywania kodu przy wystąpieniu konkretnego wyjątku. Dodanie tego rodzaju punktu wstrzymania wykonuje się poprzez przycisk na pasku narzędzi:



Można zdefiniować punkt wstrzymania zarówno przy wyjątkach przechwytywanych jak i nie przechwytywanych.

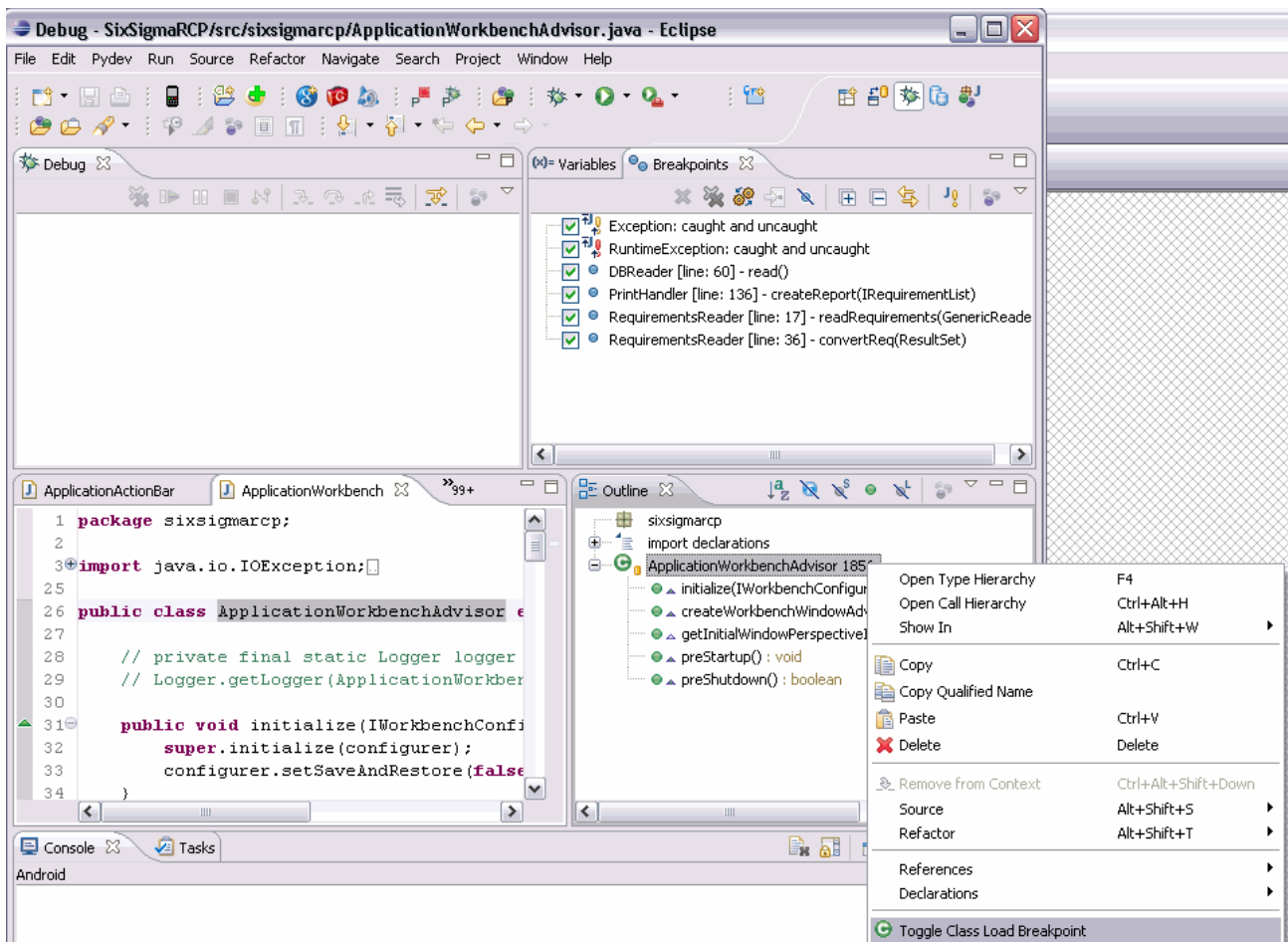
4.5. Punkt wstrzymania przy wywołaniu metody (Method Breakpoint).

Ustawianie wstrzymania dokonuje się również poprzez podwójne kliknięcie przy nagłówku metody którą chcemy obserwować. Można zdefiniować czy wykonanie ma być wstrzymane przy wywoływaniu metody czy przy powracaniu do funkcji wywołującej.



4.6. Punkt wstrzymania przy ładowaniu klasy (Class breakpoint).

Tego rodzaju wstrzymanie jest wykonywane przy załadowaniu konkretnej klasy. Aby dodać należy kliknąć prawym przyciskiem myszy na nazwie klasy w zakładce Outline i wybrać opcję "Toggle Class Load Breakpoint".



4.7. Liczba wykonań (Hit Count)

Dla każdego punktu wstrzymania można ustawić właściwość liczby wykonań. Kiedy właściwość jest ustawiona wykonanie kodu zostanie wstrzymane w momencie osiągnięcia ustawionej wartości wykonań danego fragmentu kodu dla którego jest ustawiony punkt wstrzymania.

4.8. Ponowne uruchamianie fragmentu kodu (Drop to frame)

Eclipse pozwala na wybór dowolnego punktu wykonywania programu (frame) ze stosu wykonywania kodu i ustawienie wirtualnej maszyny javy do ponownego rozpoczęcia wykonywania kodu dla wybranego punktu.

Pozwala to na wielokrotne uruchamianie części napisanego kodu.

UWAGA!!! Zmienne nie są resetowane. Jeśli wartości zmiennych zostały zmodyfikowane to podczas ponownego wykonywania żądanego fragmentu kodu, program będzie korzystał z zmodyfikowanych wartości zmiennych lub wartości zawartych w bazie danych itp.

Aby skorzystać z tej funkcji należy wybrać żądane wywołanie w stosie oraz skorzystać z przycisku na pasku narzędzi:

Debug [X]

- Main [Java Application]
 - de.vogella.debug.first.Main at localhost:64341
 - Thread [main] (Suspended (breakpoint at line 11 in Counter))
 - Counter.count() line: 11
 - Main.main(String[]) line: 12
 - C:\Programy\Java\jre6\bin\javaw.exe (05-10-2011 09:44:52)

Counter.java [X] Main.java

```
package de.vogella.debug.first;

public class Counter {
    private int result=0;
    public int getResult() {
        return result;
    }

    public void count() {
        for (int i = 0; i < 100; i++) {
            result += i + 1;
        }
    }
}
```