

# Pemrograman Berorientasi Obyek

---

ARRAY OF OBJECTS

DEBUGGING PADA ECLIPSE

# Konsep Dasar Array

---

- Java memiliki struktur data array yang dapat menyimpan sekumpulan elemen yang berukuran tetap dan memiliki tipe yang sama
- Contoh: untuk menyimpan data nilai dari 5 siswa, kita tidak perlu membuat 5 variable bertipe integer, tetapi cukup 1 variable array yang bertipe integer

```
int nilai1;  
int nilai2;  
int nilai3;  
int nilai4;  
int nilai5;
```



```
int[] nilai = new int[5];
```

# Memproses Elemen dalam Array

- Untuk memproses array, kita dapat menggunakan looping `for` atau `foreach` (khusus Java versi 1.5 ke atas)
- Berikut contoh dengan looping `for`:

Output:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

```
public class TestArray {

    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }
        // Summing all elements
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
            total += myList[i];
        }
        System.out.println("Total is " + total);
        // Finding the largest element
        double max = myList[0];
        for (int i = 1; i < myList.length; i++) {
            if (myList[i] > max) max = myList[i];
        }
        System.out.println("Max is " + max);
    }
}
```

# Memproses Elemen dalam Array

---

- Berikut contoh dengan foreach:

```
public class TestArray {  
  
    public static void main(String[] args) {  
        double[] myList = {1.9, 2.9, 3.4, 3.5};  
  
        // Print all the array elements  
        for (double element: myList) {  
            System.out.println(element);  
        }  
    }  
}
```

Output:

```
1.9  
2.9  
3.4  
3.5
```

# Memberikan Array sebagai parameter ke dalam Method

---

- Array dapat dijadikan sebagai parameter input ke dalam sebuah method

- Contoh: 

```
public static void printArray(int[] array) {  
    for (int i = 0; i < array.length; i++) {  
        System.out.print(array[i] + " ");  
    }  
}
```

- Cara menggunakan method yang memiliki array sebagai parameter inputnya:

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

atau

```
int[] data = new int[]{3,1,2,6,4,2};  
printArray(data);
```

# Mengembalikan Array sebagai parameter output dari Method

---

- Sebuah method dapat mengembalikan Array sebagai *return value*-nya. Contoh:

```
public static int[] reverse(int[] list) {  
    int[] result = new int[list.length];  
  
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {  
        result[j] = list[i];  
    }  
    return result;  
}
```

# Array of Object

---

- Variabel array tidak hanya dapat menyimpan data dengan tipe data primitif saja (int, double, float, dan char), tetapi juga dapat untuk menyimpan objek dari semua tipe data bentukan dari user (class)
- Contoh:

```
Song s1 = new Song("Hello", "Adele", 300);  
Song s2 = new Song("Roar", "Katy", 350);
```



```
Song[] s = new Song[2];  
s[0] = new Song("Hello", "Adele", 300);  
s[1] = new Song("Roar", "Katy", 350);
```

- Semua hal yang dapat dilakukan pada array biasa (seperti pengaksesan elemen, sebagai parameter input ataupun output) dapat juga dilakukan pada *array of object*

# Class Arrays pada Java

---

- Java memiliki class bawaan `Arrays` yang memiliki beberapa method statis untuk mengurutkan dan mencari data pada array, membandingkan array, dan mengisi elemen array

| No. | Perintah   | Keterangan  |
|-----|--|---|
| 1.  | <code>public static int binarySearch(Object[] a, Object key);</code> | Mencari <code>key</code> di dalam array of object <code>a</code> (yang sudah terurut) dengan teknik Binary Search. Return value: indeks dari <code>key</code> pada array  |
| 2.  | <code>public static boolean equals(long[] a, long[] a2);</code>      | Mengembalikan nilai <code>true</code> jika kedua array <code>a</code> dan <code>a2</code> adalah sama (baik dari hal nilai maupun posisi). Dapat dilakukan pada tipe data primitif yang lain ( <code>byte</code> , <code>short</code> , <code>int</code> , dll) |
| 3.  | <code>public static void fill(int[] a, int val);</code>              | Mengisikan nilai <code>val</code> ke setiap elemen pada array <code>a</code> . Dapat dilakukan pada tipe data primitif yang lain ( <code>byte</code> , <code>short</code> , <code>int</code> , dll)   |
| 4.  | <code>public static sort(Object[] a);</code>                         | Mengurutkan array <code>a</code> secara menaik. Dapat dilakukan pada tipe data primitif yang lain ( <code>byte</code> , <code>short</code> , <code>int</code> , dll)  |



# Debugging

---

- **To debug** (kata kerja): Mengidentifikasi dan menghilangkan kesalahan/error dari hardware atau software computer
- **Debugging** (kata benda): Proses mengidentifikasi dan menghilangkan kesalahan/error dari hardware atau software computer
- Beberapa teknik debugging:
  - **Interactive debugging**: Menjalankan program sambil mengecek source code dan status variable pada saat eksekusi program
  - **Print debugging**: Mengecek output yang dicetak yang mengindikasikan alur eksekusi dari sebuah proses
  - **Remote debugging**: Debugging pada program yang berjalan di luar system orang yang melakukan debug
  - **Post-mortem debugging**: Debugging pada program setelah kesalahan terjadi, misal dengan menganalisis *memory dump* dari proses yang menyebabkan error
  - **Algoritma “Wolf fence”**: Mempersempit area untuk menemukan lokasi di mana error terjadi. Analogi dengan mencari seekor serigala di hutan Alaska

# Debugging di Java dengan Eclipse

---

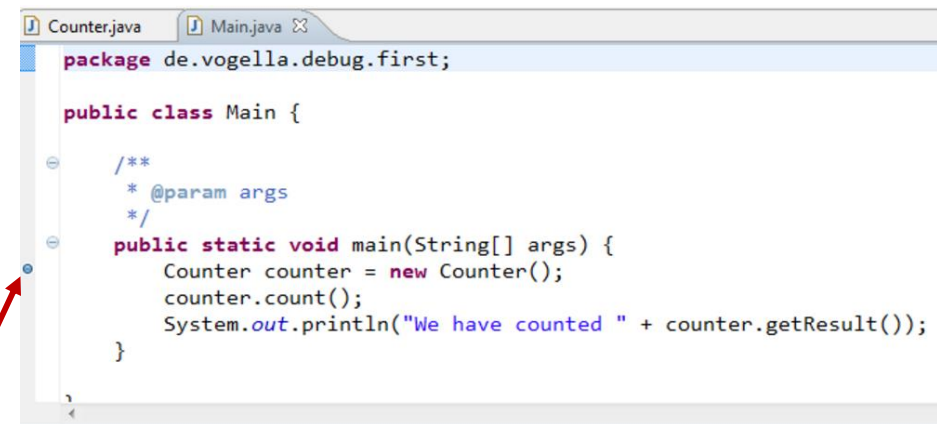
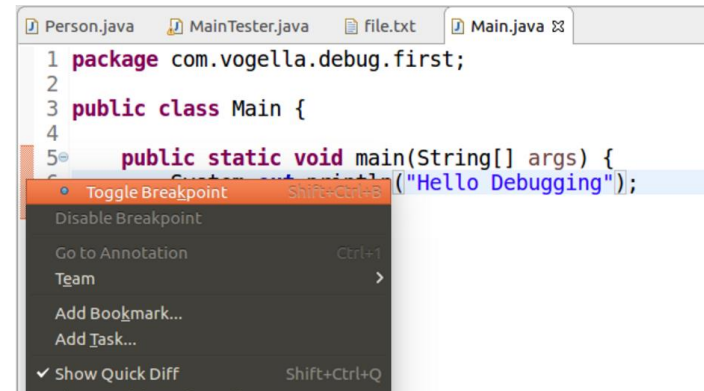
- Eclipse menyediakan fasilitas untuk debugging dengan menggunakan mode *Debug*
- Terdapat perspektif Debug yang memungkinkan kita untuk melakukan pra-konfigurasi view yang ingin diketahui nilainya dan mengontrol alur eksekusi melalui sejumlah perintah debug
- **Breakpoint:** Posisi pada source code dimana proses eksekusi akan berhenti selama proses debugging. Ketika program sudah berhenti sementara, kita dapat menginvestigasi variable, mengubah isi variable, dan lain-lain
- **Watchpoint:** Breakpoint khusus yang menghentikan proses eksekusi ketika nilai sebuah ekspresi atau variable berubah, tanpa menspesifikasikan dimana posisi kemunculannya

# 1. Menetapkan Breakpoint

- ❑ Klik kanan di margin kiri pada editor Java dan pilih *Toggle Breakpoint*
- ❑ Cara lain, dengan langsung double click pada posisi yang dipilih
- ❑ Pada contoh gambar kedua di samping, kita menetapkan breakpoint pada baris


```
Counter counter = new Counter();
```

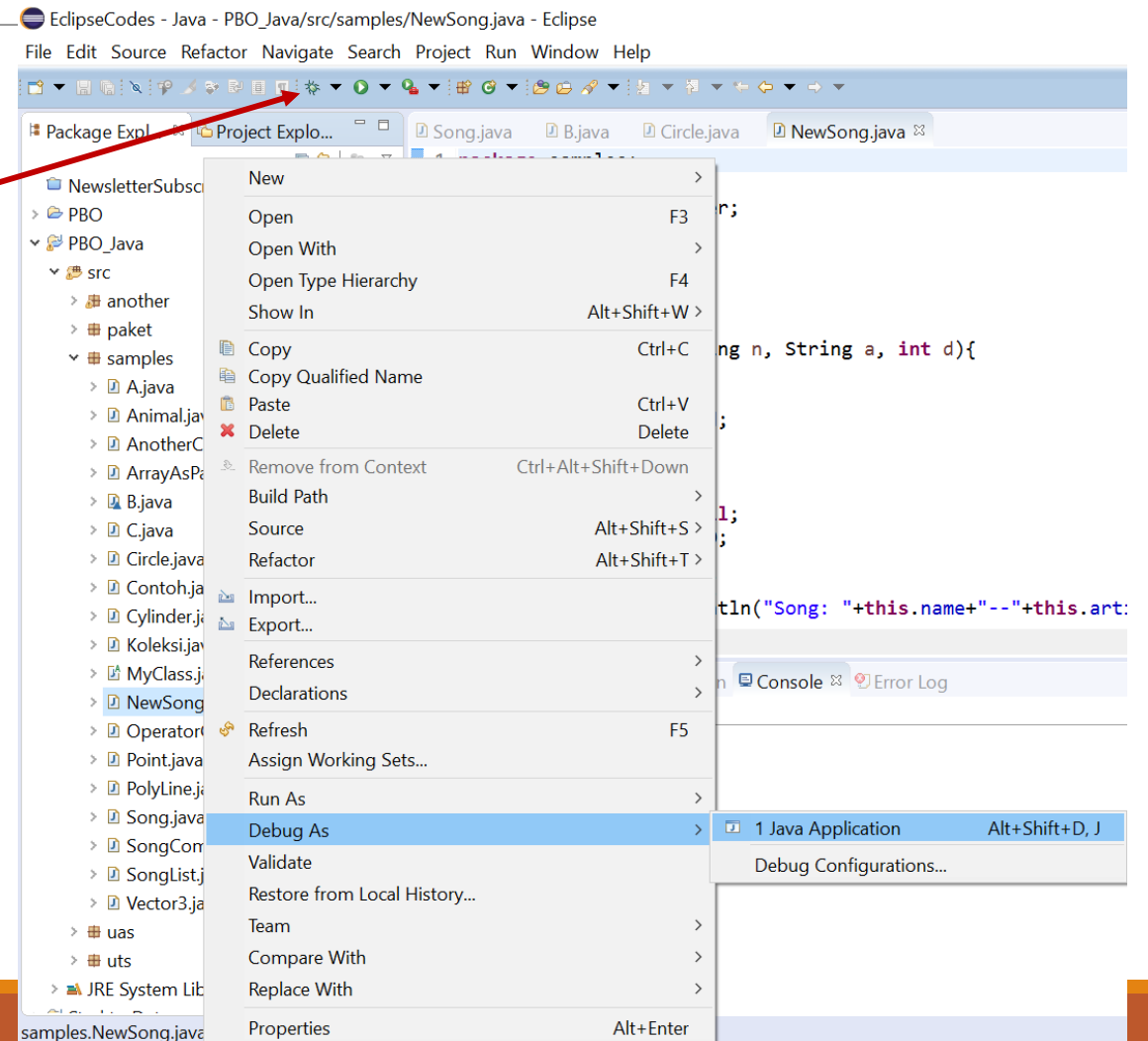
- ❑ Breakpoint memiliki sejumlah *properties* yang dapat diatur untuk membatasi aktivasi dari breakpoint. Misal: Hanya akan aktif setelah iterasi ke-12 atau kita dapat menuliskan sebuah ekspresi kondisi. Eksekusi program hanya berhenti pada breakpoint jika kondisi bernilai benar



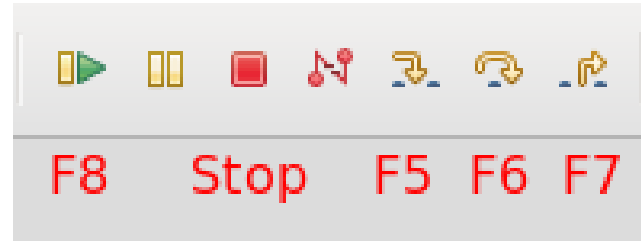
Tanda breakpoint

## 2. Memulai Debugger

- Untuk melakukan debugging pada aplikasi, pilih file Java yang memiliki method *main*. Klik kanan di Package Explorer dan pilih Debug As → Java Application
- Alternatif lain, klik tombol Debug di toolbar atas 
- Jika breakpoint belum didefinisikan, program akan berjalan seperti biasa
- Eclipse akan mengkonfirmasi untuk berpindah ke perspektif Debug ketika sebuah stop point dicapai

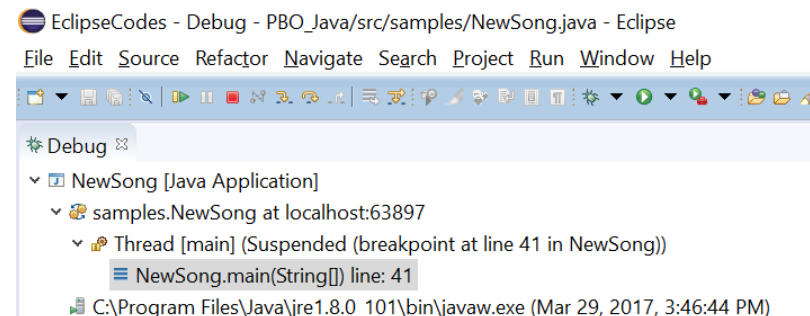


### 3. Mengontrol Eksekusi Program




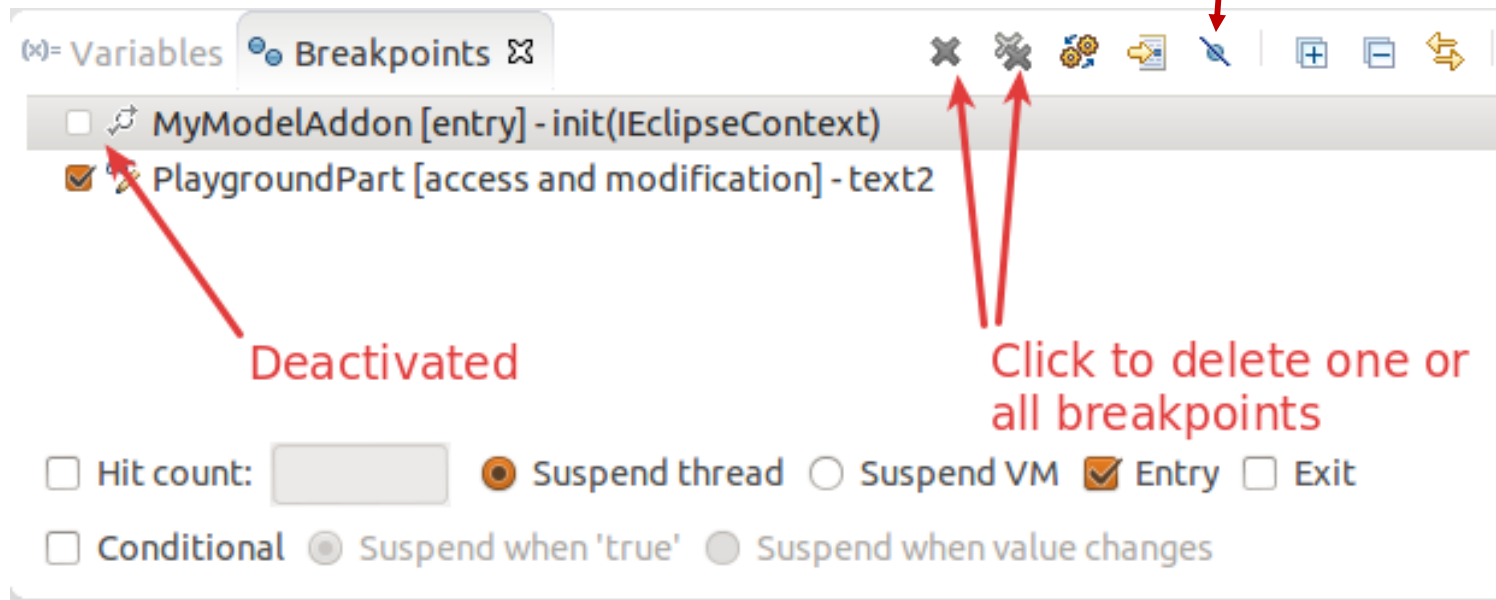
| Key | Fungsi  |
|-----|---|
| F5  | <b>[STEP INTO]</b> F5 mengeksekusi baris kode yang sedang terpilih dan akan bergerak menuju baris berikutnya. Jika baris kode yang terpilih adalah pemanggilan ke sebuah method, debugger akan masuk ke dalam method tersebut |
| F6  | <b>[STEP OVER]</b> F6 akan tetap mengeksekusi method tetapi melewati penelusuran ke dalam method (Tidak masuk ke dalam method)  |
| F7  | <b>[STEP RETURN]</b> F7 keluar dari dalam method yang sedang dieksekusi dan kembali ke pemanggilnya   |
| F8  | <b>[RESUME]</b> F8 menyuruh untuk melanjutkan eksekusi program hingga breakpoint atau watchpoint berikutnya tercapai  |

Call Stack menunjukkan bagian program yang sekarang sedang dieksekusi dan relasinya dengan bagian lain dari program. Stack yang sedang aktif ditampilkan di jendela *Debug*



## 4. Jendela Breakpoint dan deaktivasi breakpoint

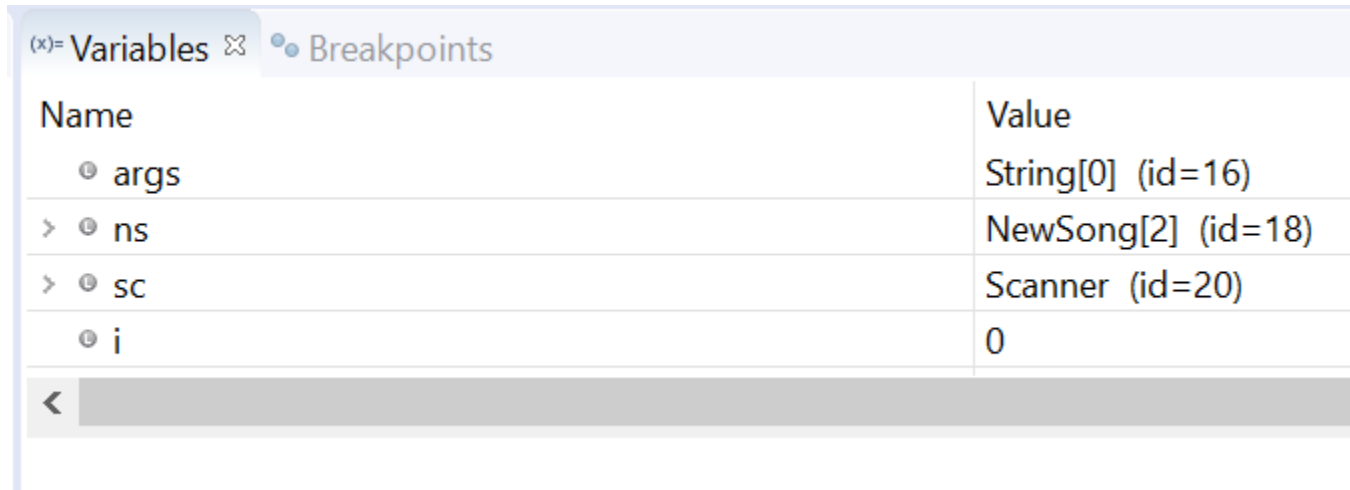
- Di jendela ini kita dapat menghapus dan men-deaktivasi *breakpoint* dan *watchpoint*
- Untuk mendeaktivasi semua breakpoint, klik tombol 



# 5. Evaluasi variable pada debugger

---

- Jendela Variabel menampilkan field dan variable lokal dari stack yang sedang dieksekusi

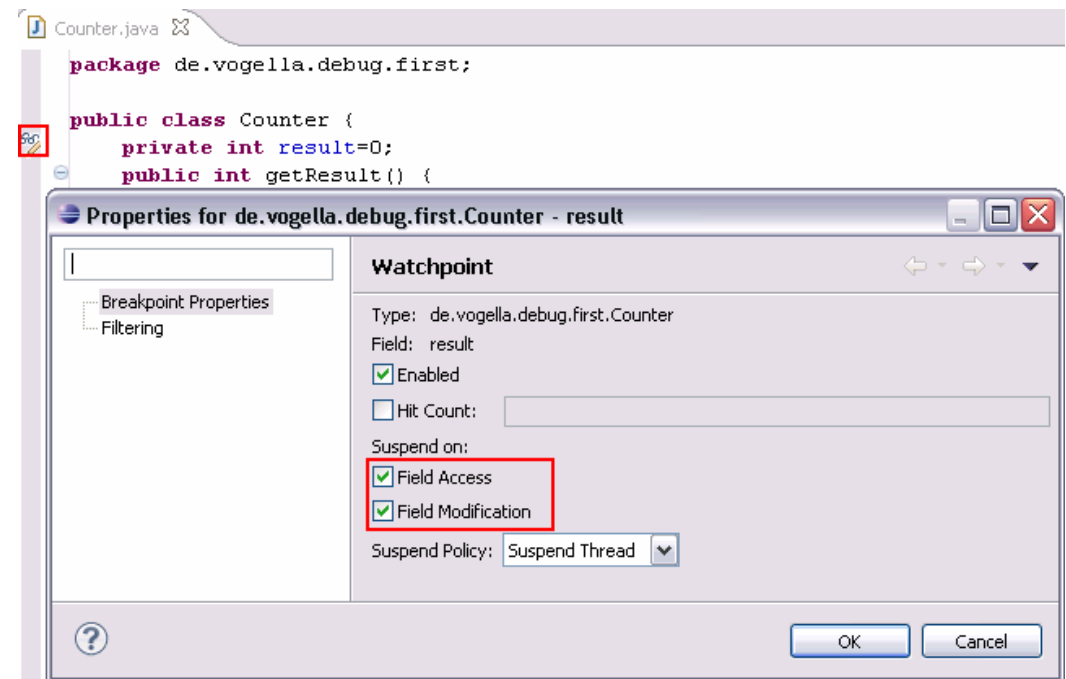


The screenshot shows a debugger interface with two tabs: 'Variables' (selected) and 'Breakpoints'. The 'Variables' tab displays a table of local variables. The table has two columns: 'Name' and 'Value'. The variables listed are 'args' (String[0] (id=16)), 'ns' (NewSong[2] (id=18)), 'sc' (Scanner (id=20)), and 'i' (0). There is a left arrow icon at the bottom left of the table.

| Name | Value              |
|------|--------------------|
| args | String[0] (id=16)  |
| > ns | NewSong[2] (id=18) |
| > sc | Scanner (id=20)    |
| i    | 0                  |

# 6. Menetapkan watchpoint

- Watchpoint adalah breakpoint yang ditetapkan pada sebuah field/variable dan debugger akan berhenti jika nilai dari variable tersebut dibaca atau diubah
- Watchpoint diset dengan klik ganda di sisi kiri, sejajar dengan deklarasi variable
- Watchpoint juga memiliki *properties* yang dapat diatur apakah eksekusi akan berhenti saat akses pembacaan variable (Field Access), pengubahan isi variable (Field Modification), atau keduanya





# Latihan Debugging

---

1. Buat project Java baru dengan nama `pbo.eclipse.debug.first`
2. Buat package baru dalam project dengan nama `pbo.eclipse.debug.first`
3. Buat 2 class berikut:

```
1 package pbo.eclipse.debug.first;
2
3 public class Counter {
4     private int result = 0;
5
6     public int getResult(){
7         return result;
8     }
9
10    public void count(){
11        for (int i=0;i<100;i++){
12            result+=i+1;
13        }
14    }
15 }
```

```
1 package pbo.eclipse.debug.first;
2
3 public class Main {
4     public static void main(String[] args){
5         Counter counter = new Counter();
6         counter.count();
7         System.out.println("Kita sudah menghitung "+counter.getResult());
8     }
9 }
```

# Latihan Debugging (2)

---

4. Tentukan breakpoint di class `Counter`. Debug program Anda dan ikuti eksekusi dari method `count`
5. Hapus breakpoint Anda dan tambahkan breakpoint baru untuk *class loading*. Debug lagi program Anda dan verifikasi bahwa debugger berhenti ketika class Anda di-load