Pengenalan GUI dengan Eclipse SWT

SWT adalah GUI berbagai platform yang dikembangkan oleh IBM. Kenapa IBM membuat sistem GUI lain dan tidak menggunakan rangka kerja GUI yang sudah ada pada Java? Untuk menjawab pertanyaan ini, kita perlu melihat kilas balik tentang Java di awal perkembangannya.

Sun telah membuat rangka kerja GUI yang dinamakan AWT (Abstract Windowing Toolkit). Rangka kerja AWT menggunakan widget (komponen pada GUI, seperti tombol, menu, teks, dll) alami yang merupakan bawaan dari setiap sistem operasi. Akan tetapi widget tersebut memiliki masalah pada LCD. Masalah LCD ini mengakibatkan hilangnya beberapa fitur pada sistem operasi lain. Atau dengan kata lain, jika platform A memiliki widget 1 - 40 dan platform B memiliki widget 20 - 25, maka rangka kerja AWT hanya bisa digunakan untuk sebagian widget yang beririsan saja.

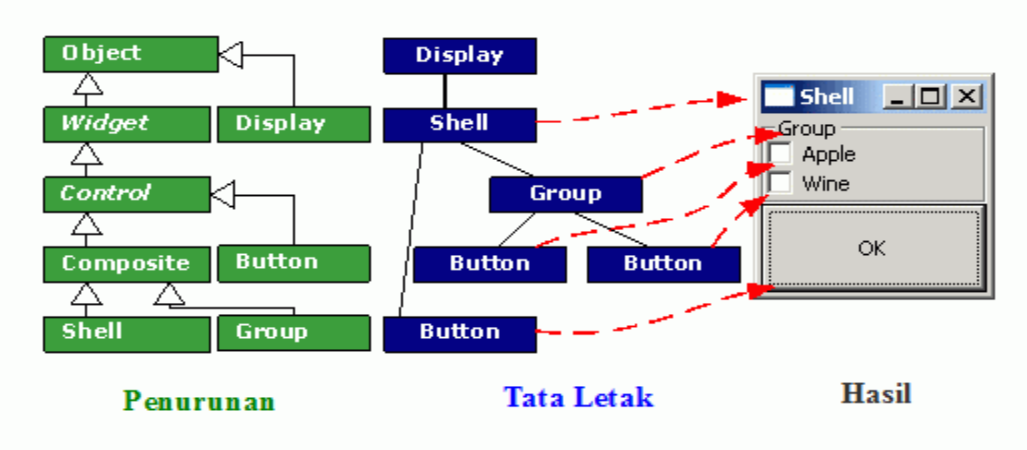
Untuk memecahkan masalah ini, Sun membuat rangka kerja baru yang merupakan emulasi widget, bukan menggunakan widget bawaan platform. Hal ini memecahkan masalah LCD dan memberikan rangka kerja yang kaya widget, akan tetapi masalah lain muncul. Misalnya, aplikasi Swing tidak lagi mirip seperti aplikasi lain pada platform di mana ia dijalankan.

Dengan adanya perbaikan pada JVM, aplikasi Swing tetap saja mengalami masalah kinerja karena sifatnya yang berupa emulasi. Hal ini tidak ditemukan pada widget bawaan platform karena widget ini lebih dekat dengan sistem operasi. Lebih jauh, aplikasi Swing menghabiskan lebih banyak memori yang tidak bisa digunakan untuk piranti kecil seperti PDA dan telepon genggam.

IBM memutuskan bahwa pendekatan tersebut tidak memenuhi kebutuhannya. Akhirnya, IBM membuat pustaka GUI baru yang disebut SWT, yang memecahkan masalah yang terdapat pada rangka kerja AWT dan Swing. Rangka kerja SWT langsung mengakses widget bawaan sistem operasi melalui JNI. Jika widget tersebut tidak tersedia, maka SWT akan mengemulasi widget yang hilang itu.

Blok Penyusun Suatu Aplikasi SWT

Display, Shell dan kumpulan Widget adalah blok penyusun aplikasi SWT. Display bertugas untuk mengatur perulangan event (dari keyboard atau mouse) dan mengatur komunikasi antara thread UI dan thread lainnya. Shell adalah jendela di mana aplikasi berjalan. Setiap aplikasi SWT memiliki paling tidak satu Display dan satu atau lebih instansi Shell.



Gambar di atas mengilustrasikan aplikasi SWT dari sudut pandang yang berbeda. Diagram pertama di sebelah kiri adalah diagram kelas-kelas turunan dari objek-objek UI. Diagram tengah adalah bagaimana objek UI diletakkan, dan gambar kanan adalah UI yang dihasilkan.

Jika suatu aplikasi menggunakan beberapa thread, setiap thread akan memiliki instansi objek Display masing-masing. Kita bisa mengambil instansi suatu objek Display dengan menggunakan metode statik `Display.getCurrent()`.

Suatu Shell melambangkan jendela suatu aplikasi. Shell bisa ditampilkan dalam ukuran seluruh layar, ukuran biasa, atau dikecilkan hingga tak terlihat. Ada dua jenis shell yang tersedia. Yang pertama adalah shell dengan tingkat paling tinggi, yaitu yang dibuat langsung sebagai anak dari jendela utama Display. Jenis kedua adalah shell dialog yang bergantung pada shell-shell lainnya.

Jenis suatu Shell bergantung pada bit gaya (style bit) yang diberikan pada konstruktornya. Nilai awalnya adalah `DialogShell`. Artinya, jika tidak ada parameter yang diberikan pada konstruktornya, maka shell yang dibuat akan bertipe `DialogShell`. Jika parameter konstruktornya berupa objek bertipe `Display`, maka ia akan menjadi shell tingkat atas.

Beberapa sifat widget harus ditentukan pada saat widget tersebut dibuat. Sifat widget tersebut disebut bit gaya (style bit). Bit gaya ini adalah konstanta yang didefinisikan dalam kelas SWT. Misalnya,

```
Button tombol = new Button(shell, bitGaya)
```

Kita bisa menggunakan lebih dari satu gaya dengan menggunakan operator OR atau `|`. Misalnya, kita ingin membuat tombol yang bisa ditekan dan memiliki garis tepi, kita bisa menggunakan `SWT.PUSH | SWT.BORDER` sebagai parameter bit gayanya.

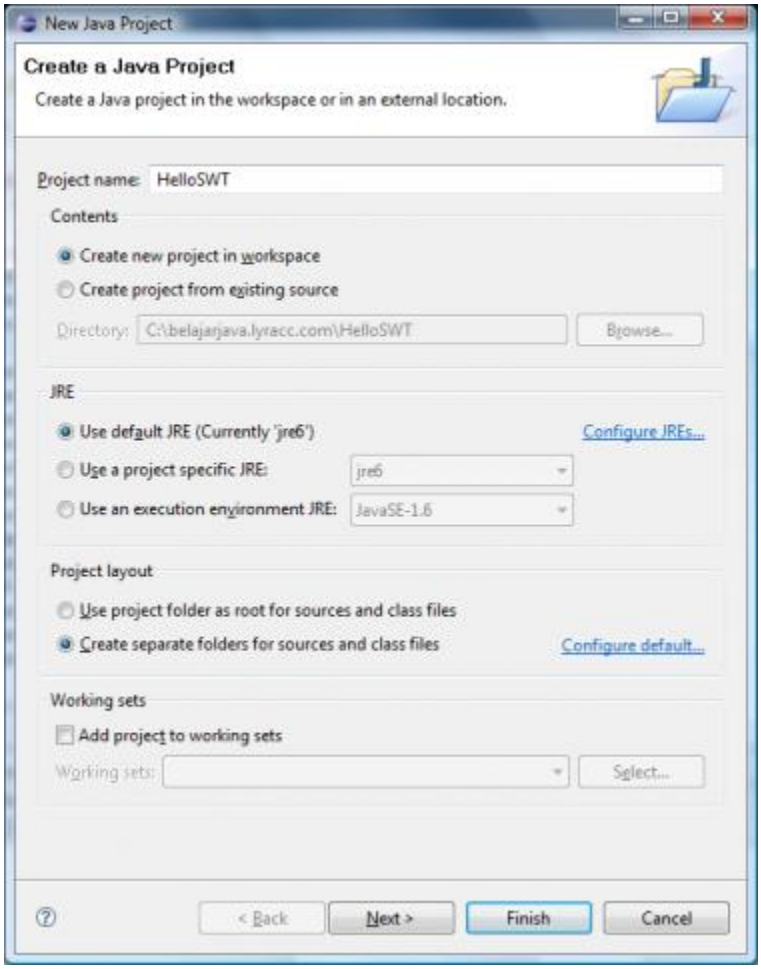
Memulai SWT

Untuk memulai pemrograman dengan SWT, mari kita buat program sederhana yang kita namakan **HelloSWT**.

- Anda membutuhkan "link" ke pustaka SWT. Ada dua cara untuk melakukannya :

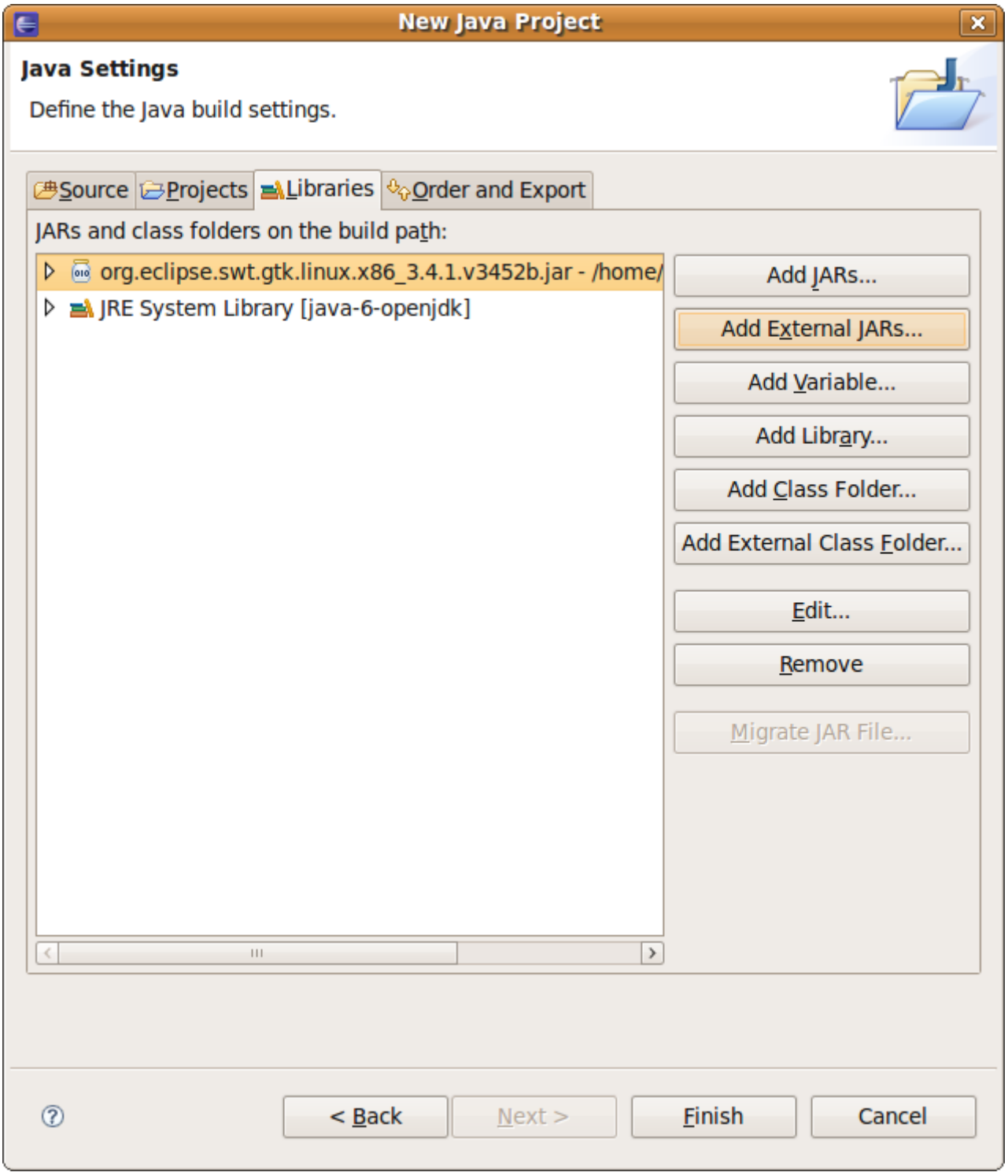
1. Menggunakan pustaka SWT bawaan dari Eclipse. Untuk ini Anda tidak perlu mendownload apa-apa. Pustaka SWT bawaan eclipse terdapat pada direktori Eclipse Anda, misalnya pada Windows (**C:\eclipse\plugins\org.eclipse.swt.win32.win32.x86_3.4.1.v3449c.jar**) atau pada Linux (**eclise/plugins/org.eclipse.swt.gtk.linux.x86_3.4.1.v3452b.jar**)
2. Menggunakan pustaka SWT beserta sumber kodenya untuk dokumentasi lebih lengkap. Pustaka SWT ini bisa diunduh pada alamat berikut : http://www.eclipse.org/swt/. Pilih Releases -> Stable -> (platform Anda, misalnya Windos atau Linux). Setelah diunduh, import ke dalam Eclipse seperti Anda mengimport proyek-proyek pada website ini, yaitu dengan **File** -> **Import** -> **General** -> **Existing Projects Into Workspace** -> **Select Archieve File** -> (file zip SWT hasil download) -> **Finish**. Langkah ini hanya dilakukan satu kali saja.

- Buat proyek baru dari Eclipse, **File** -> **New** -> **Java Project**, kemudian isi Project Name dengan **HelloSWT** dan click **Next**

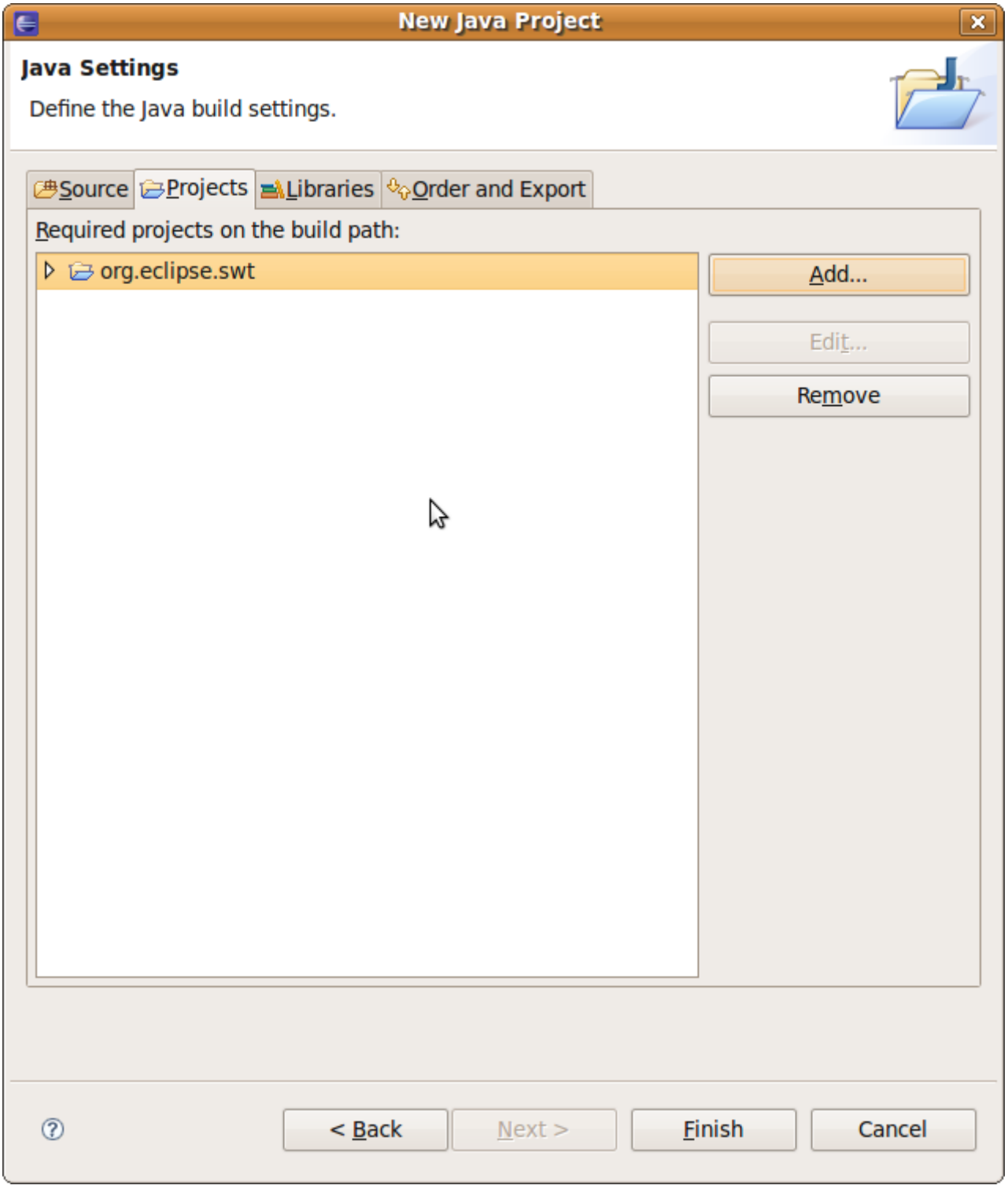


- Kita harus menambahkan pustaka SWT ke dalam proyek kita**.**

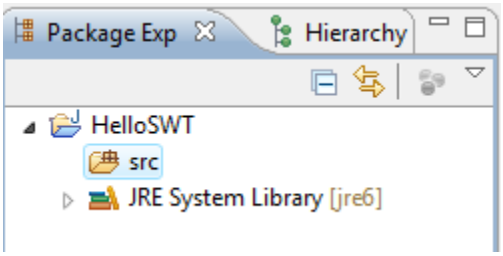
1. Jika Anda menggunakan SWT bawaan Eclipse seperti di atas, pilih halaman **Libraries**. Klik **Add External JAR**, kemudian navigasi ke direktori plugin di dalam instalasi Eclipse Anda. Misalnya di komputer saya, direktori itu berada di **C:\eclipse\plugins**. Pilih JAR untuk SWT pada direktori tersebut. Namanya tergantung dari sistem operasi Anda, misalnya pada Windows file ini bernama `org.eclipse.swt.win32.win32.x86_3.4.1.v3449c`. Klik **OK** setelah file ini ditambahkan di dalam folder Libraries.



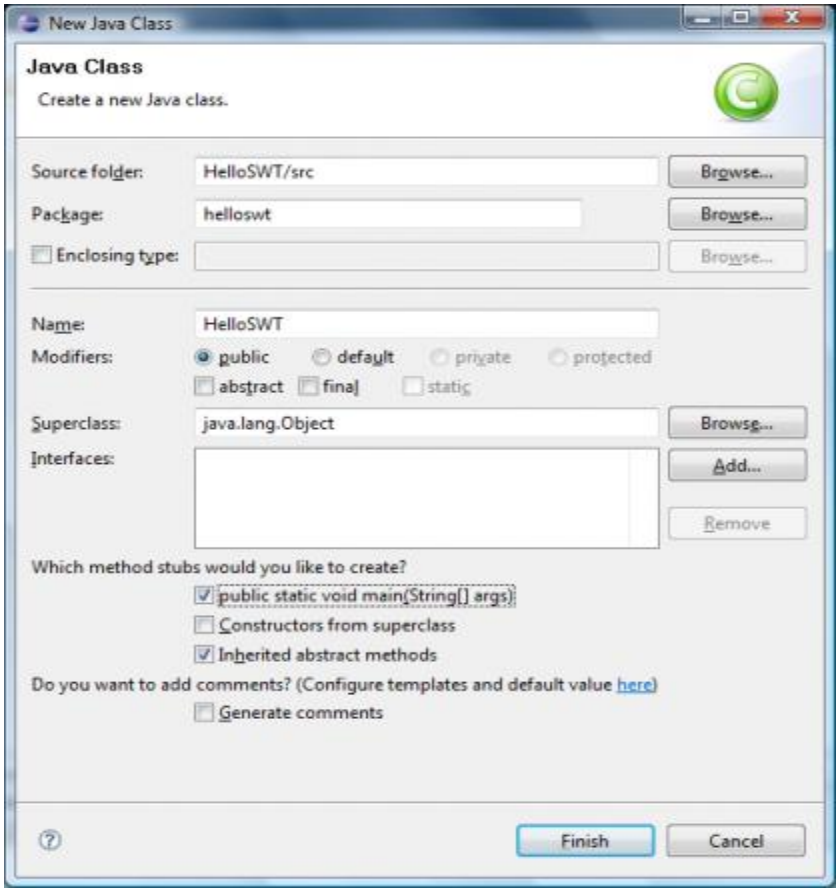
1. Jika Anda mengunduh proyek SWT, pilih halaman **Projects**. Klik **Add**, kemudian tambahkan org.eclipse.swt. Kemudian klik OK setelah proyek ini ditambahkan.



- Folder proyek baru akan dibuat, seperti pada gambar berikut.



- Kemudian **klik kanan** pada folder src, dan pilih **New** -> **Class**. Isi nama kelas dengan **HelloSWT**, nama package dengan **helloswt**, dan tik "public static void main(String[] args)" untuk membuat metode `main()` secara otomatis.

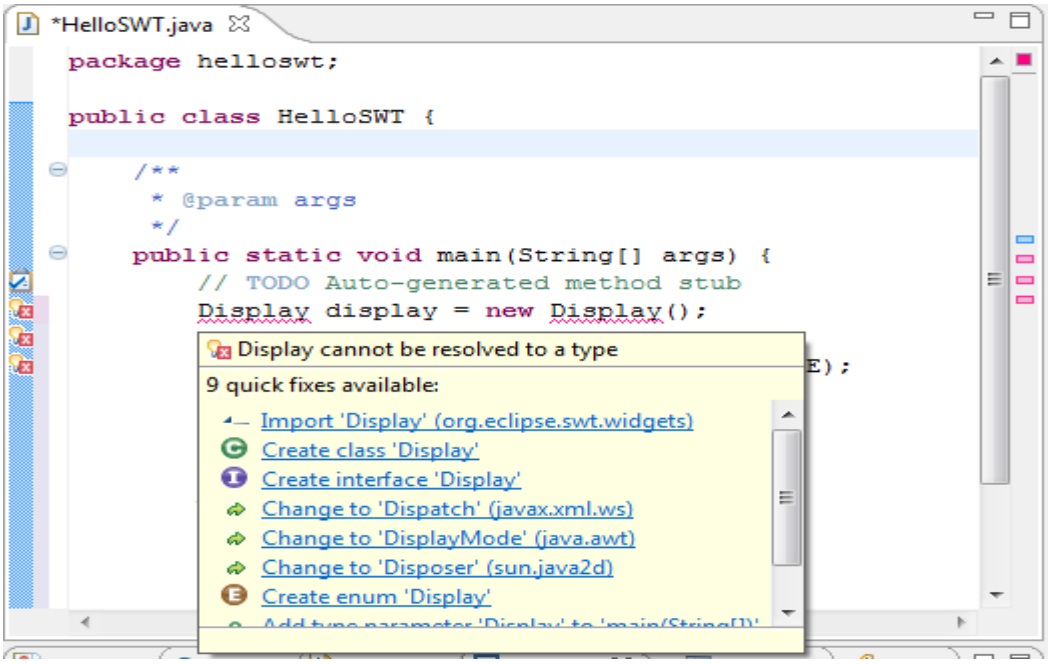


- Ketik kode berikut ini di dalam metode `main()`

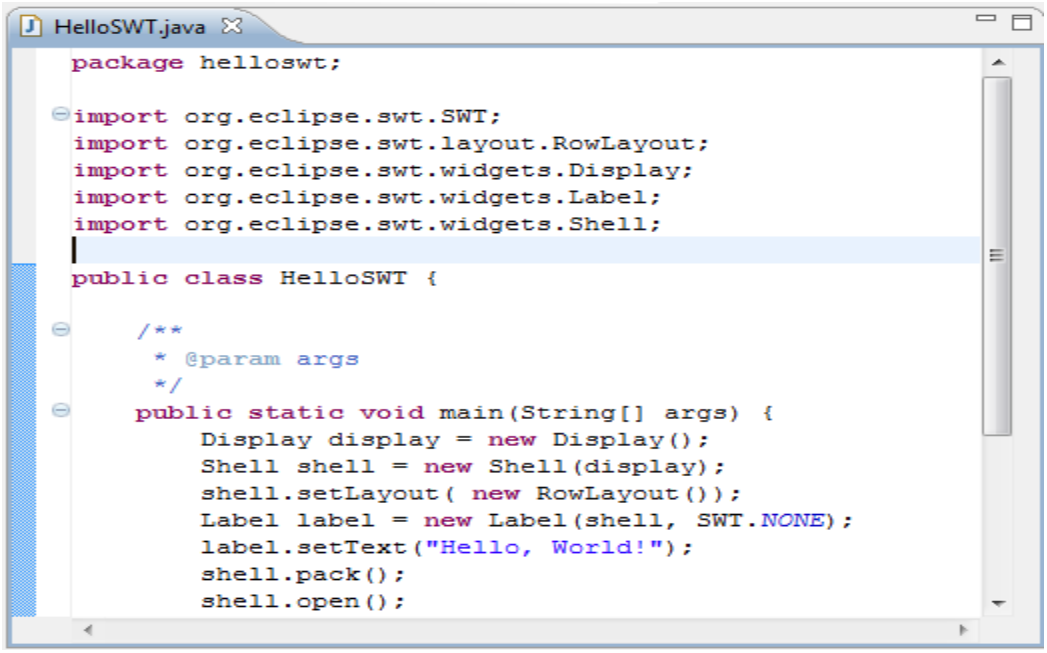
```
Display display = new Display();
Shell shell = new Shell(display);
shell.setLayout( new RowLayout());
Label label = new Label(shell, SWT.NONE);
label.setText("Hello, World!");
shell.pack();
shell.open();
while (!shell.isDisposed())
    if (!display.readAndDispatch())
        display.sleep();
display.dispose();
label.dispose();
```

- Perhatikan bahwa `Display`, `Shell`, `RowLayout` dan `Label` diberi tanda merah sebagai tanda bahwa kesalahan program terjadi.

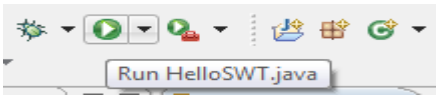
- Sekarang pindahkan mouse Anda pada kesalahan pada kelas `Display`. Eclipse akan memberi tahu Anda kesalahan apa yang terjadi. Pada contoh ini, kelas '`Display`' belum diketahui oleh Eclipse. Klik pada pilihan **Import 'Display' (org.eclipse.swt.widgets)**. Lihat sekarang di awal program Anda Eclipse menambahkan **`import`** `org.eclipse.swt.widgets.Display;` secara otomatis.



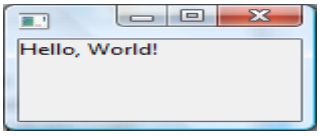
- Lakukan hal yang sama pada semua kesalahan, yaitu `Shell`, `Label`, `RowLayout` dan `SWT.NONE`, sehingga tidak ada lagi kesalahan yang dilaporkan oleh Eclipse.



- Untuk menjalankan program Anda, klik tombol **Run** seperti pada gambar berikut.



- Berikut ini adalah tampilan program SWT pertama Anda.

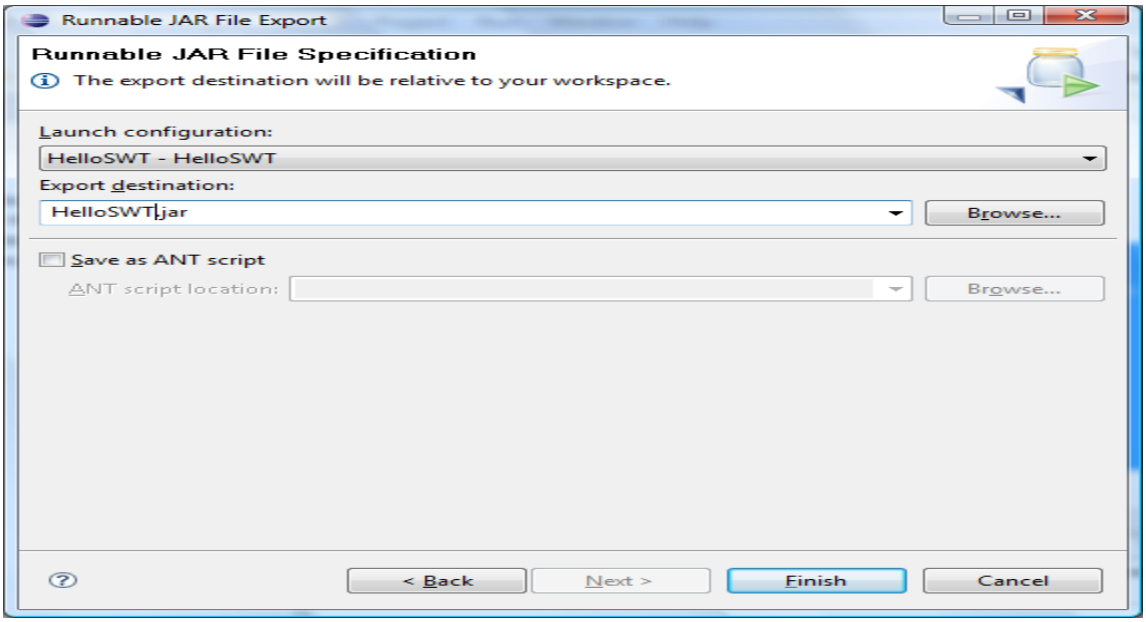


Mengkompilasi dan Membuat Program SWT Berjalan di Luar Eclipse

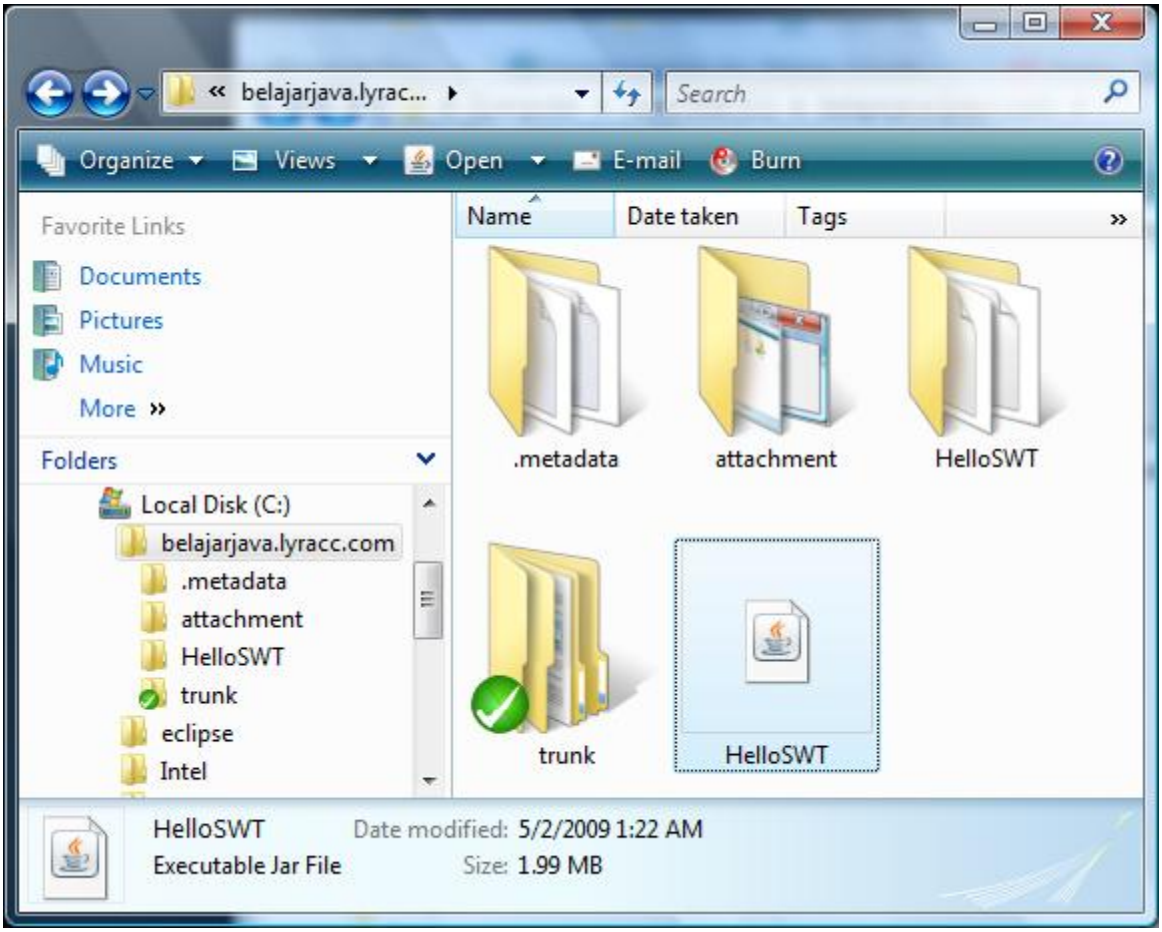
Untuk bisa membuat program kita berguna untuk orang lain, kita harus bisa mengkompilasi dan membuat program tersebut sebagai satu "paket". Proses ini disebut menyebarkan (deploy). Pada Eclipse, program yang telah kita buat bisa kita deploy dalam format JAR (Java Archive), yang menggabungkan semua file kelas yang sudah dikompilasi beserta pustaka yang dibutuhkan untuk bisa menjalankan program tersebut.

Mari kita lihat contoh program **HelloSWT** yang sudah dibahas sebelumnya. Untuk men-deploy program ini sehingga bisa dijalankan di luar Eclipse, lakukan langkah-langkah berikut :

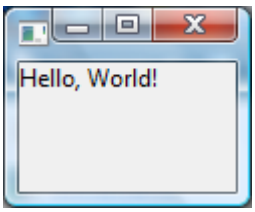
- **Klik kanan** pada nama proyek Anda, kemudian pilih **Export**. Setelah itu kotak dialog akan terbuka. Kemudian pilih **Runnable JAR File** dari dalam folder **Java**.
- Pilih **HelloSWT - HelloSWT** pada **Launch configuration** dan juga masukkan di mana file JAR hasilnya akan ditempatkan. Jika nama direktori tidak diberikan, Eclipse akan meletakkan file JAR tersebut pada direktori di mana workspace berada, yaitu di mana data proyek ditempatkan. (Misalnya, di komputer saya proyek HelloSWT berada pada direktori C:\belajarjava.lyracc.com\HelloSWT, maka file JAR tersebut akan diletakkan pada direktori C:\belajarjava.lyracc.com\)



- Klik **Finish**, kemudian **OK** untuk setiap pertanyaan yang mungkin muncul berikutnya. Sekarang, gunakan Windows Explorer untuk mencari di mana file JAR hasilnya berada. **Double click** pada file tersebut dan Anda akan bisa menjalankannya langsung seperti aplikasi lainnya. Anda bisa juga mengirimkan file ini melalui email atau mengkopinya melalui flash disk ke teman atau kerabat Anda tanpa perlu menyertakan Eclipse.



- **CATATAN** : Ingat bahwa SWT bergantung penuh pada sistem operasi yang Anda gunakan, karena SWT menggunakan widget bawaan sistem operasi seperti dijelaskan pada bagian sebelumnya. Ketika Anda membuat program ini pertama kali, Anda menambahkan pustaka SWT yang spesifik terhadap sistem operasi di mana Eclipse berjalan. Artinya, jika Anda menambahkan pustaka **org.eclipse.swt.win32.win32.x86_3.4.1.v3449c**, maka program JAR Anda hanya akan bisa dijalankan pada sistem operasi Windows. Untuk bisa menjalankannya pada sistem operasi lain, misalnya Linux, maka Anda harus menambahkan pustaka SWT khusus Linux yang bisa diunduh secara terpisah atau menggunakan Eclipse pada Linux.
- Berikut ini adalah screen shot hasil jalannya program



Melihat Lebih Dekat HelloSWT

Secara umum aplikasi SWT membutuhkan beberapa langkah sebagai berikut :

1. Buat Display baru
2. Buat satu atau lebih Shell
3. Buat manager layout untuk Shell baru
4. Buat widget di dalam shell
5. Buka jendela shell
6. Buat perulangan pengirim event
7. Buang (dispose) display dan widget-widget lainnya
8. Tentunya import berbagai paket yang diperlukan oleh program

```
package helloswt;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.RowLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;

public class HelloSWT {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout( new RowLayout());
        Label label = new Label(shell, SWT.NONE);
        label.setText("Hello, World!");
        shell.pack();
        shell.open();
        while (!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
        label.dispose();
    }
}
```

Mari kita bahas satu per satu

1. Membuat Display baru

Membuat Display dapat dilakukan dengan membuat instansi dari objek bertipe Display, misalnya seperti perintah berikut

```
Display display = new Display();
```

Display (tampilan) adalah objek maya yang merupakan kontainer induk semua komponen GUI. Suatu tampilan sebetulnya tidak terlihat, akan tetapi komponen yang ditambahkan ke dalam tampilan dapat dilihat. Biasanya, suatu aplikasi hanya membutuhkan satu tampilan.

2. Membuat Shell baru

Membuat Shell dapat dilakukan dengan membuat instansi dari objek bertipe Shell, misalnya seperti perintah berikut

```
Shell shell = new Shell(display);
```

Pada pernyataan di atas, konstruktor `Shell` mengambil parameter yang merupakan induk di mana shell ini akan diletakkan. Pada pernyataan di atas, kita membuat objek `shell` bertipe `Shell` yang memiliki induk objek `display`.

Shell adalah jendela suatu aplikasi. Shell tertinggi adalah shell yang berada langsung di bawah objek bertipe Display, dan merupakan jendela utama suatu aplikasi. Shell sebenarnya adalah widget komposit, yaitu suatu widget yang terdiri dari beberapa widget lain. Oleh karenanya shell juga bisa berisi shell lain. Untuk aplikasi sederhana yang menampilkan kata "Hello World!", kita hanya membutuhkan satu shell saja yang kita letakkan pada objek display.

3. Membuat manager layout untuk Shell baru

Untuk membuat manager layout, kita menggunakan metode instansi dari objek shell, yaitu `setLayout()` yang mengambil parameter jenis layout yang akan disusun pada shell. Misalnya,

```
shell.setLayout( new RowLayout());
```

Pada contoh di atas, kita memerintahkan objek shell untuk mengatur widget-widget di dalamnya dalam urutan seperti baris. `RowLayout` adalah kelas layout yang digunakan objek shell untuk mengatur objek widget di dalamnya.

Manager layout digunakan untuk mengatur secara otomatis peletakkan setiap widget. Pada SWT, objek komposit seperti Shell tidak mengetahui bagaimana widget harus ditampilkan. Oleh karenanya jika kita tidak memberikan manager layout kepada objek bertipe Shell, maka tidak ada widget yang akan ditampilkan.

Bahasan khusus tentang layout cukup kompleks, dan karenanya kita akan bahas tersendiri pada bagian selanjutnya.

4. Membuat widget di dalam shell

Widget adalah komponen pembangun GUI. Menu, tombol, teks, kotak cek, kotak input teks, hingga kanvas, adalah contoh-contoh widget. Setiap widget memiliki sifat dan cara kerja yang berbeda-beda. Pada contoh ini, kita menggunakan widget yang sangat sederhana yang dinamakan Label. Widget ini digunakan untuk menampilkan teks, misalnya digunakan sebagai label atau pertanyaan pada suatu input teks.

Pada contoh HelloSWT, kita gunakan konstruktor yang mengambil 2 parameter. Yang pertama, adalah kontainer induk di mana Label ini akan ditampilkan, dan parameter kedua adalah bit gaya.

```
Label label = new Label(shell, SWT.NONE);
```

Label ini kemudian kita isi dengan kata "Hello Word!" untuk menampilkan kata ini pada jendela utama.

```
label.setText("Hello, World!");
```

5. Membuka jendela shell

Setelah Display, Shell, dan widget-widgetnya disiapkan, kita perlu secara eksplisit memerintahkan shell untuk membuka jendela dan menggambar seluruh widget yang sudah disiapkan. Untuk ini kita gunakan perintah

```
shell.open();
```

6. Membuat perulangan pengirim event

SWT mengharuskan kita untuk membuat sendiri perulangan untuk mengirimkan event. Artinya, blok perintah berikut ini "harus" selalu ada dalam semua aplikasi SWT yang akan kita buat.

```
while (!shell.isDisposed())
    if (!display.readAndDispatch())
        display.sleep();
```

7. Membuang (dispose) display dan widget-widget lainnya

Karena SWT bekerja langsung dengan widget bawaan sistem operasi, maka tanggung jawab untuk menghapus komponen GUI ada di tangan programmernya. Hal ini disebabkan widget tersebut berada di luar jangkauan pemulung memori Java yang hanya bisa meraih objek yang dibuat dan dijalankan dalam JVM (Java Virtual Machine).

Untuk menghapus widget dari memori, kita dapat menggunakan metode `dispose()` yang dimiliki oleh hampir semua widget SWT. Pada contoh program ini, kita memiliki 2 widget, yaitu shell dan label, sehingga keduanya harus dihapus dengan menggunakan perintah

```
display.dispose();
label.dispose();
```

8. Mengimport berbagai paket yang diperlukan oleh program

Kita bisa menggunakan Eclipse untuk membantu kita mengimpor setiap paket (seperti yang diilustrasikan pada bagian sebelumnya), atau menggunakan wildcard seperti `import org.eclipse.swt.widgets.*`

Beberapa paket penting yang tersedia pada SWT :

Nama Paket	Kegunaan
org.eclipse.swt	Berisi kelas SWT, kumpulan konstanta, dan kelas-kelas pengecualian dan kesalahan
org.eclipse.swt.widgets	Widget dan kelas-kelas yang berhubungan dengannya, misalnya menu, kotak dialog, dukungan event, kelas super abstrak untuk layout, dan sebagainya
org.eclipse.swt.events	Event tingkat tinggi, pendengar event, dan colokan (adapter)

org.eclipse.swt.layout	Kelas layout standar seperti FillLayout, RowLayout, GridLayout, FormLayout, dan kelas-kelas lainnya
org.eclipse.swt.graphics	Huruf, warna, gambar, dan operasi grafik dasar, seperti untuk menggambar garis dan lingkaran
org.eclipse.swt.custom	Widget buatan sendiri dan kelas lain yang dibuat khusus untuk Eclipse; tidak bergantung pada platform apapun (implementasinya sama pada semua sistem operasi)
org.eclipse.swt.dnd	Kelas bantuan untuk melakukan drag-and-drop pada berbagai platform
org.eclipse.swt.accessibility	Perluasan untuk memberikan bantuan dan dukungan untuk orang cacat
org.eclipse.swt.printing	Dukugan printer dan kotak dialog untuk melakukan percetakan
org.eclipse.swt.program	Berisi kelas Program, untuk mempermudah pada saat program dijalankan, pemetaan dokumen, dan icon; juga dibuat untuk Eclipse, akan tetapi sangat bergantung pada platformnya (implementasinya bisa berbeda untuk setiap sistem operasi)
org.eclipse.swt.browser	Menyediakan widget sebagai web broswer, dan kelas-kelas yang mendukungnya. Menggunakan browser bawaan sistem operasi untuk implementasinya
org.eclipse.swt.awt	Menyediakan dukungan untuk membuat UI yang berisi campuran widget dari SWT dan AWT
org.eclipse.swt.win32.ole	Hanya tersedia pada Windows; menyediakan fasilitas untuk mengakses OLE (Object Linking and Embedding)

Pengenalan Konsep Widget pada SWT

Secara tradisional, widget bisa dibayangkan seperti perangkat abstrak yang berguna untuk melakukan tugas tertentu. Istilah ini popules dalam bidan ekonomi. Para pembuat software meminjam istilah ini untuk menyebut suatu paket program pembuat GUI (grahical user interface). SWT adalah kepanjangan dari Standard Widget Toolkit karena widget merupakan dasar pembangun suatu aplikasi yang dibangun dengan menggunakan SWT.

Apa itu widget?

Widget adalah elemen GUI yang dapat berinteraksi dengan user. Widget mengatur dan menentukan kondisinya sendiri dengan memggunakan kombinasi beberapa operasi grafik. Dengan menggunakan mouse atau keyboard, user bisa mengubah kondisi suatu widget. Ketika kondisi suatu widget berubah, baik diubah oleh user ataupun diubah oleh suatu kode program, maka widget akan menggambar dirinya kembali untuk merefleksikan hasil perubahannya.

Siklus hidup widget

Widget memiliki siklus hidup sendiri. Widget dibuat oleh programmer dan dibuang ketika tidak lagi dibutuhkan. Karena siklus hidup suatu widget sangat penting untuk memahami SWT, maka kita akan bahas lebih jauh pada bagian ini.

1. Membuat Widget

Widget dibuat dengan menggunakan konstruktornya, atau dengan kata lain membuat instansi suatu kelas. Ketika widget dibuat, widget akan mengambil sumber daya komputer (memori, prosesor) dari sistem operasi. Artinya semua widget akan berada di level sistem operasi, sehingga unjuk kerja dan pengaturan memori akan lebih optimal.

Konstruktor akan mengambil argumen yang biasanya tidak akan bisa diubah setelah widget dibuat. Ada 4 kemungkinan konstruktor dari suatu jenis widget :

- `Widget()`
- `Widget(Widget induk)`
- `Widget(Widget induk, int bitGaya)`
- `Widget(Widget induk, int bitGaya, int posisi)`

Widget tidak bisa dibuat tanpa induk. Ini bisa dilihat dari pengenalan tentang widget di bagian sebelumnya bagaimana widget tersusun secara hirarkis. Jenis induknya tergantung dari jenis widgetnya. Misalnya, induk dari suatu menu harus berupa menu, dan tidak bisa berupa tombol. Kompiler akan mengecek apakah induk suatu widget sesuai dengan tipenya, jika tidak, kompiler akan menampilkan pesan kesalahan. (Pada beberapa platform, kita dibolehkan untuk mengubah induk suatu widget. Akan tetapi SWT tidak memiliki metode getParent() pada kelas-kelas Widgetnya. Ini disebabkan karena kompleksitasnya yang mengharuskan kita untuk melakukan type cast kepada tipe induk yang benar.)

Bit gaya (style) adalah nilai bit integer yang digunakan untuk mengatur perilaku dan tampilan suatu widget. Biasanya bit gaya hanya dibutuhkan pada saat widget dibuat, misalnya memilih jenis editor multi baris atau baris tunggal. Karena atribut ini tidak bisa diubah setelah widget dibuat, maka gaya suatu widget juga tidak bisa diubah setelah widget diinstansikan.

Semua gaya widget dikumpulkan sebagai konstanta pada kelas `org.eclipse.swt.SWT`. Kita bisa menggambungkan beberapa gaya dengan menggunakan operasi OR. Misalnya kode berikut akan membuat widget teks multi baris yang memiliki tepi serta scroll bar horizontal dan vertikal.

```
Text teks = new Text (induk, SWT.MULTI | SWT.V_SCROLL | SWT.H_SCROLL | SWT.BORDER);
```

Gaya suatu widget bisa kita ambil setelah widget tersebut dibuat dengan menggunakan metode instansi `getStyle()`. Untuk mengujinya, kita bisa menggunakan operasi AND yang jika hasilnya tidak nol, maka widget tersebut memiliki gaya yang diuji. Misalnya, kode berikut menguji apakah suatu widget teks memiliki tepi, kemudian mencetak hasilnya pada konsol.

```
if ((teks.getStyle() & SWT.BORDER) != 0)
    System.out.println("teks memiliki tepi");
else
    System.out.println("teks tidak memiliki tepi");
```

Paremeter posisi memungkinkan kita membuat widget baru pada suatu induk di posisi tertentu. Jika tidak diberikan, maka secara otomatis widget baru akan ditempatkan di akhir posisi. Hal ini akan lebih memudahkan kita untuk membuat widget pada urutan tertentu, misalnya membuat menu dengan urutan tertentu.

2. Membuang widget

Ketika widget tidak lagi dibutuhkan, termasuk ketika program selesai, maka widget harus dihapus secara eksplisit dengan menggunakan metode instansi `dispose()`. Metode ini akan menyembunyikan widget, menghapus widget yang ditampungnya, dan membuat semua referensi di dalamnya menjadi **`null`**.

Jika metode `dispose()` tidak dipanggil setelah program selesai dijalankan, maka hal ini akan menyebabkan kebocoran memori, di mana memori pada sistem operasi lambat laun akan habis dipenuhi oleh widget yang tidak dihapus setelah program selesai.

Memanggil metode `dispose()` untuk widget yang sudah dibuang, akan melemparkan pengecualian `SWTException`. Untuk mengetahui apakah suatu widget sudah dihapus, kita bisa menggunakan metode instansi `isDisposed()`.

Ada dua aturan yang penting untuk diingat :

- Jika Anda membuat widget, maka pastikan Anda menghapusnya. Atau dengan kata lain, semua widget yang Anda buat dengan konstruktor wajib dihapus. Akan tetapi jika Anda mendapatkan widget dari metode instansi suatu widget, misalnya `Font huruf = kontrol.getFont()`, maka Anda tidak perlu menghapusnya
- Jika Anda menghapus suatu widget, maka semua widget di bawahnya akan dihapus juga. Atau dengan kata lain, jika Anda menghapus suatu Shell, maka isi seluruh Shell akan dihapus secara otomatis. Demikian juga halnya dengan menu, jika Anda menghapus menu utama, maka semua sub-menu akan otomatis dihapus.

Event dan Listener

Event

Event adalah inti dari pemrograman GUI. GUI program tidak memiliki alur apa yang akan terjadi ketika program dijalankan, dalam arti langkah per langkah dari awal hingga akhir. Akan tetapi, program harus bisa bereaksi akan berbagai event (kejadian) yang bisa berasal dari mouse atau keyboard. User bisa menekan tombol keyboard apa saja, menggeser mouse, atau menekan tombol mouse. User bisa melakukannya kapan saja, dan komputer harus bisa bereaksi dengan tepat.

Dalam Java, event dilambangkan oleh objek. Ketika suatu event terjadi, sistem akan mengumpulkan informasi yang sesuai dengan even tersebut, kemudian membuat objek yang berisi informasi ini. Jenis even berbeda dilambangkan oleh objek dengan kelas yang berbeda pula. Setelah objek event dibuat, ia akan diberikan sebagai parameter pada subrutin yang ditugaskan untuk menangani event tersebut. Dengan menulis subrutin ini, programmer bisa memberi tahu apa yang harus dilakukan jika mouse diklik atau keyboard ditekan, misalnya.

Sebagai programmer Java, kita hanya melihat even dari sisi yang lebih umum. Banyak sekali hal yang terjadi antara tombol ditekan hingga subrutin yang kita buat melakukan tugasnya. Secara garis besar, dalam metode `main()` kita harus menuliskan perulangan dalam bentuk seperti :

```
ketika program masih berjalan:
```

```
    Tunggu hingga even berikutnya terjadi
    Panggil subrutin untuk menangai event tersebut
```

Perulangan ini disebut perulangan event. Pada SWT kita harus menulis sendiri perulangan event ini pada metode `main()` kita dalam bentuk

```
while (!shell.isDisposed())
    if (!display.readAndDispatch())
        display.sleep();
```

SWT memiliki dua jenis event, yaitu **event tanpa tipe** dan event **bertipe.**

Event tanpa tipe diwakili oleh kelas `Event`. Kelas ini menyimpan informasi yang berbeda-beda tergantung dari jenis. Akan tetapi secara umum semua event jenis apapun memiliki informasi berikut :

Informasi	**Penjelasan**
display	Display di mana event ini terjadi
widget	Widget di mana event ini terjadi
type	Jenis event yang terjadi

Jenis event yang terjadi beserta penjelasannya dirangkum dalam tabel berikut ini :

Jenis Event	**Penjelasan**
SWT.KeyDown	Tombol ditekan
SWT.KeyUp	Tombol dilepaskan
SWT.MouseDown	Tombol mouse ditekan
SWT.MouseUp	Tombol mouse dilepaskan
SWT.MouseMove	Mouse berpindah posisi
SWT.MouseEnter	Mouse masuk ke wilayah klien
SWT.MouseHover	Mouse berada di sekitar klien
SWT.MouseExit	Mouse keluar dari wilayah klien
SWT.MouseDoubleClick	Mouse di-double click
SWT.Paint	Suatu widget diberi perintah untuk menggambar dirinya
SWT.Move	Posisi suatu widget berubah
SWT.Resize	Ukuran widget berubah
SWT.Dispose	Widget dihapus
SWT.Selection	Suatu aksi pemilihan dilakukan dalam widget (misalnya memilih item pada drop down list
SWT.DefaultSelection	Ada pilihan awal pada suatu widget
SWT.FocusIn	Widget menerima fokus dari keyboard
SWT.FocusOut	Widget kehilangan fokus dari keyboard
SWT.Expand	Item pada pohon dikembangkan

SWT.Collapse	Item pada pohon ditutup
SWT.Iconify	Jendela Shell diminimasi
SWT.Deiconify	Jendela Shell dibuka (restore)
SWT.Close	Jendela Shell ditutup (dengan tombol X)
SWT.Show	Widget bisa dilihat
SWT.Hide	Widget disembunyikan
SWT.Modify	Teks berubah pada suatu kontrol
SWT.Verify	Input teks sedang diverifikasi
SWT.Activate	Widget sedang diaktifkan
SWT.Deactivate	Widget dimatikan
SWT.Help	User meminta pertolongan tentang suatu widget
SWT.DragDetect	Aksi drag-and-drop dideteksi
SWT.MenuDetect	User menekan tombol kik kanan untuk mengaktifkan menu konteks
SWT.Arm	Item pada menu digambarkan
SWT.Traverse	Navigasi pada keyboard dideteksi
SWT.HardKeyDown	Tombol pada hardware ditekan (untuk perangkat genggam)
SWT.HardKeyUp	Tombol pada hardware dilepas (untuk perangkat genggam)

Suatu event juga bisa memiliki tipe. Artinya event yang dihasilkan merupakan objek bertipe suatu kelas, misalnya MouseEvent, bukan hanya bertipe kelas Event yang jenisnya disimpan dalam variabel tertentu. Event bertipe mengikuti pola JavaBeans standar. Kelas-kelas ini terdapat dalam paket `org.eclipse.swt.events`

Tabel berikut menggambarkan perbandingan antara event bertipe dan jenis event dari event tanpa tipe.

Event Bertipe	**Jenis Event Tanpa Tipe**
ArmEvent	SWT.Arm
ControlEvent	SWT.Move SWT.Resize
DisposeEvent	SWT.Dispose
FocusEvent	SWT.FocusIn SWT.FocusOut
HelpEvent	SWT.Help
KeyEvent	SWT.KeyDown SWT.KeyUp
MenuEvent	SWT.Hide

	SWT.Show
ModifyEvent	SWT.Modify
MouseEvent	SWT.MouseDoubleClick SWT.MouseDown SWT.MouseUp
MouseEvent	SWT.MouseMove
MouseEvent	SWT.MouseEnter SWT.MouseExit SWT.MouseHover
PaintEvent	SWT.Paint
SelectionEvent	SWT.DefaultSelection SWT.Selection
ShellEvent	SWT.Activate SWT.Close SWT.Deactivate SWT.Iconify SWT.Deiconify
TraverseEvent	SWT.Traverse
TreeEvent	SWT.Collapse SWT.Expand
VerifyEvent	SWT.Verify

Pertanyannya kenapa ada dua jenis event yang berbeda?

Pada versi awal SWT, hanya ada satu jenis yaitu jenis tanpa tipe. Setelah diskusi yang cukup antar beberapa programmer Eclipse, komunitas user SWT dan developernya, maka akhirnya diputuskan untuk menambahkan jenis bertipe seperti pada JavaBeans. Alasannya adalah untuk memudahkan pemrograman SWT bagi programmer yang sudah terlanjur terbiasa dengan AWT/Swing. Jenis tanpa tipe masih tetap ada seperti biasa.

Listener

Supaya suatu event berarti, suatu program harus bisa mendeteksi event dan bereaksi akan event tersebut. Untuk mendeteksi suatu event, suatu program harus mendengarkannya. Mendengarkan event ini dilakukan oleh objek yang berna,a pendengar event (event listener). Objek listener harus memiliki

metode instansi untuk menangani event yang didengarkannya. Bentuknya bervariasi tergantung dari jenis event yang ditanganinya.

Ada beberapa hal detail yang harus diingat untuk bisa bekerja dengan event. Beberapa langkah yang harus diingat :

1. Menambahkan import paket yang dibutuhkan, misalnya "org.eclipse.swt.events"

2. Mendeklarasikan kelas yang mengimplementasikan interface suatu listener

3. Menambahkan aksi yang dilakukan oleh kelas baru tersebut. Aksi ini adalah aksi yang dilakukan untuk menangani suatu event

4. Mendaftarkan event tersebut ke komponen yang mungkin memberikan event.

Objek apapun bisa bertindah sebagai event listener asalkan ia mengimplementasikan interface yang tepat. Suatu komponen dapat mendengarkan event yang dihasilkannya sendiri. Suatu kelas dapat dibuat secara khusus hanya untuk mendengarkan suatu event. Kebanyakan orang menganggap lebih mudah untuk menggunakan kelas bertingkat anonim untuk mendengarkan suatu objek. (Kelas bertingkat anonim telah dibahas sebelumnya di sini).

Seperti hanya Event, SWT memiliki dua jenis listener : tanpa tipe dan bertipe. Masing-masing digunakan untuk menangani event tanpa tipe dan event bertipe.

Listener Tanpa Tipe (untuk menangani event tanpa tipe)

nterface generik yang digunakan untuk menangai event tanpa tipe dinamakan `Listener`. Listener tanpa tipe dapat ditambahkan pada suatu widget dengan menggunakan metode `addListener()`.

Metode `addListener()` memiliki bentuk seperti

```
addListener(int jenisEvent, Listener listener)
```

Metode ini akan menambah `listener` ke dalam koleksi listener yang akan dipanggil ketika event tipe tertentu terjadi. Ketika suatu event terjadi, maka listener akan dipanggil dengan menggunakan metode `handleEvent()`.

Contoh berikut mengilustrasikan bagaimana menambah suatu Listener ke dalam suatu widget, yang akan dipanggil ketika event SWT.Dispose terjadi.

```
widget.addListener(SWT.Dispose, new Listener() {
    public void handleEvent(Event event) {
        // widget was disposed
    }
});
```

Perhatikan bahwa bentuk di atas menggunakan kelas anonim yang langsung diturunkan dari interface Listener. Di dalam kelas anonim tersebut, metode `handleEvent()` harus diimplementasikan, di mana implementasinya adalah tugas yang harus dilakukan ketika widget dihapus.

Jika beberapa listener ditambahkan, maka mereka akan dipanggil dengan urutan ketika mereka ditambahkan (first in first called). Artinya listener pertama diberi kesempatan untuk mengolah event (mungkin untuk memfilter sejumlah data) sebelum listener lain melakukan tugasnya. Kita juga bisa menambah listener yang sama beberapa kali, yang artinya listener tersebut akan dipanggil beberapa kali.

Kita bisa menghapus listener dari suatu widget dengan menggunakan metode `removeListener()`. Bentuknya adalah sebagai berikut

```
removeListener(int jenisEvent, Listener listener)
```

Untuk menghapus suatu listener, kita harus memberikan instansi yang persisi sama ketika listener tersebut ditambahkan. JIka beberapa instansi listener yang sama sudah ditambahkan sebelumnya, maka kita harus menghapusnya berulang kali sejumlah ia ditambahkan. Secara umum, menghapus suatu listener mungkin tidak diperlukan. Listener akan diambil oleh pemulung memori ketika suatu widget dihapus dan listener ini tidak lagi digunakan di manapun di dalam program.

Kita juga bisa memanggil suatu listener untuk melakukan tugas tertentu, yaitu dengan menggunakan metode `notifyListeners`, yang memiliki bentuk seperti

```
notifyListeners(int jenisEvent, Event event)
```

`jenisEvent` adalah jenis event yang akan dilakukan, dan Event adalah objek yang berisi event yang akan dilakukan. Hal penting yang harus dicatat adalah memanggil metode `notifyListeners()` tidak menyebabkan event akan benar-benar terjadi. Misalnya memanggil `notifyListeners()` dengan jenis `SWT.MouseDown` tidak akan menyebabkan tombol terlihat seperti ditekan. Dan juga memanggil `notifyListeners()` tidak menjamin bahwa semua data pada event terinisialisasi seperti pada event sesungguhnya.

Listener Bertipe (untuk menangani Event bertipe)

Karena event bertipe dibuat mengikuti pola listener pada JavaBeans, penggunaan listener bertipe sangat mirip dengan penggunaan listener pada AWT/Swing.

Misalnya untuk mendengarkan event ketika suatu widget dihapus, maka aplikasi bisa menggunakan `addDisposeListener()`, yang memiliki bentuk seperti

```
addDisposeListener(DisposeListener listener)
```

Metode ini menambahkan listener ke dalam koleksi listener yang akan dipanggil ketike suatu widget dihapus. Ketika suatu widget dihapus, maka listener akan dipanggil dengan memanggil metode `widgetDisposed()`.

Potongan kode berikut digunakan untuk menangani event ketika suatu widget dihapus.

```
widget.addDisposeListener(new DisposeListener() {
    public void widgetDisposed(DisposeEvent event) {
        // widget was disposed
    }
});
```

`DisposeListener` adalah suatu interface. Jika ada lebih dari satu metode yang didefinisikan dalam listener, maka SWT akan secara otomatis menambahkan kelas adapter yang berisi implementasi kosong dari metode-metode yang didefinisikan pada interface tersebut. Misalnya interface `SelectionListener` memiliki dua metode, yaitu `widgetSelected()` dan `widgetDefaultSelected`, yang keduanya mengambil argumen bertipe `SelectionEvents`. Dalam hal ini kelas `SelectionAdapter` tersedia untuk memberikan implementasi kosong untuk setiap metode. Kelas adapter ini hanyalah sebagai kelas bantu yang sesuai dengan konvensi pada listener JavaBeans.

Listener bertipe bisa dihapus dengan menggunakan metode penghapus untuk setiap listner. Misalnya, listener untuk event ketika suatu widget dihapus bisa dibuang dengan menggunakan `removeDisposeListener()`, yang memiliki bentuk

```
removeDisposeListener(DisposeListener listener)
```

`listener` adalah objek listener yang akan dihapus dari koleksi listener yang dipanggil ketika widget dihapus.

Tabel berikut merangkum event bertipe, nama interface listener yang menanganinya, serta metode pada interface tersebut, dibandingkan dengan listener tanpa tipe.

Event Bertipe	**Interface Listener**	**Metode**	**Jenis Event Tanpa Tipe**
ArmEvent	ArmListener	widgetArmed(ArmEvent)	SWT.Arm
ControlEvent	ControlListener (dan ControlAdapter)	controlMoved(ControlEvent) controlResized(ControlEvent)	SWT.Move SWT.Resize
DisposeEvent	DisposeListener	widgetDisposed(DisposeEvent)	SWT.Dispose
FocusEvent	FocusListener (dan FocusAdapter)	focusGained(FocusEvent) focusLost(FocusEvent)	SWT.FocusIn SWT.FocusOut
HelpEvent	HelpListener	helpRequested(HelpEvent)	SWT.Help
KeyEvent	KeyListener (dan KeyAdapter)	keyPressed(KeyEvent) keyReleased(keyEvent)	SWT.KeyDown SWT.KeyUp
MenuEvent	MenuListener (dan MenuAdapter)	menuHidden(MenuEvent) menuShown(MenuEvent)	SWT.Hide SWT.Show
ModifyEvent	ModifyListener	modifyText(ModifyEvent)	SWT.Modify
MouseEvent	MouseListener (dan MouseAdapter)	mouseDoubleClick(MouseEvent) mouseDown(MouseEvent) mouseUp(MouseEvent)	SWT.MouseDoubleClick SWT.MouseDown SWT.MouseUp
MouseEvent	MouseMoveListener	mouseMove(MouseEvent)	SWT.MouseMove

MouseEvent	MouseTrackListener (dan MouseTrackAdapter)	mouseEnter(MouseEvent) mouseExit(MouseEvent) mouseHover(MouseEvent)	SWT.MouseEnter SWT.MouseExit SWT.MouseHover
PaintEvent	PaintListener	paintControl(PaintEvent)	SWT.Paint
SelectionEvent	SelectionListener (dan SelectionAdapter)	widgetDefaultSelected(SelectionEvent) widgetSelected(SelectionEvent)	SWT.DefaultSelection SWT.Selection
ShellEvent	ShallListener (dan ShellAdapter)	shellActivated(ShellEvent) shellClosed(ShellEvent) shellDeactivated(ShellEvent) shellIconified(ShellEvent) shellDeiconified(ShellEvent)	SWT.Activate SWT.Close SWT.Deactivate SWT.Iconify SWT.Deiconify
TraverseEvent	TraverseListener	keyTraversed(TraverseEvent)	SWT.Traverse
TreeEvent	TreeListener (dan TreeAdapter)	treeCollapsed(TreeEvent) treeExpanded(TreeEvent)	SWT.Collapse SWT.Expand
VerifyEvent	VerifyListener	verifyText(VerifyEvent)	SWT.Verify

Penanganan Mouse

- Semua sistem operasi di mana SWT diimplementasikan mendukung perangkat tunjuk. Biasanya berbentuk mouse, akan tetapi bisa jadi berupa trackball, trackpad, atau jenis perangkat keras lainnya. Pada komputer genggam, perangkat tunjuk bisa jadi berupa stylus. Untuk mempermudah pembahasan, kita akan gunakan mouse sebagai perangkat tunjuk, tidak peduli bagaimana perangkat aslinya.
- Posisi suatu mouse biasanya digambarkan dalam bentuk ikon kecil pada layar yang disebut kursor. Hal ini berlaku untuk semua platform, kecuali pada Windows CE, karena perangkat Windows CE biasanya berupa perangkat tunjuk "langsung", seperti stylys, yang tidak membutuhkan kursor.
- Mouse biasanya memiliki tiga tombol (kecuali pada Macintosh yang hanya memiliki satu tombol, walaupun sebenarnya mouse lebih dari 1 tombol pun bisa digunakan). Mouse digunakan untuk menunjuk, klik, geser (drag) dan memilih komponen kontrol GUI. Bisa juga digunakan untuk menampilkan menu konteks yang biasanya ditampilkan dengan mengklik kanan suatu mouse. Perilaku "drag-and-drop" mouse kurang lebih sangat bergantung pada platformnya.
- Ketika kita menggeser mouse, kursor akan berbah bentuk, tergantung dari kontrol apa di bawahnya. Misalnya, widget teks akan mengubah tampilan kursor seperti huruf I untuk

menunjukkan bahwa user bisa mengetikkan sesuatu pada widget tersebut. Di dalam kursor, ada titik pusat yang menunjukkan koordinat x dan y suatu mouse ketika event pada mouse terjadi.

- **Event pada Mouse**
- Ketika tombol mouse ditekan atau mouse digerakkan, event mouse dibuat dan akan diberikan kepada widget yang ada di bawahnya. Akan tetapi ketika tombol mouse ditekan dan ditahan (terus ditekan), dan mouse berada di luar widget (mungkin ada di widget lain atau pada aplikasi lain di desktop), eventnya akan diberikan kepada widget awal di mana mouse tersebut ditekan. Pengalihan event sementara ini disebut pengambilan mouse. Widget yang menerima event disebut widget pengambil. Pengambilan mouse terhadi secara otomatis pada SWT. (Ini mungkin bukan sesuatu masalah, akan tetapi sebagai informasi saja kepada Anda).
- Tabel - Isi Event Mouse ketika suatu tombol mouse ditekan, dilepaskan atau mouse digeser

Nama Field	Penjelasan
Button	Tombol yang ditekan atau dilepaskan
X	Koordinat x ketika event terjadi
Y	Koordinat y ketika event terjadi
stateMask	Bit mask yang menyatakan kondisi keyboard dan mouse sebelum event terjadi

- Ketika mouse ditekan atau dilepas, field bernama "button" akan diisi oleh tombol mana yang ditekan. Tombol mouse diberi nomor dari kiri ke kanan yang dimulai dari 1. Untuk user kidal (dan mengkonfigurasi sistem operasi untuk orang kidal), penomoran tombol tetap sama, akan tetapi dimulai dari kanan ke kiri. Pemetaan tombol untuk orang kidal ini tidak tampak oleh SWT dan aplikasi kita, karena dilakukan secara otomatis oleh sistem operasi.
- Ketika terjadi event pada mouse, koordinat x dan y-nya juga dilaporkan dalam event. Koordinat yang dilaporkan adalah koordinat relatif widget ketika event tersebut dibuat (bukan koordinat global layar atau aplikasi kita). Karena user mungkin telah memindahkan mouse setelah menekan tombol, maka lokasi sebenarnya ketika event ini ditangani mungkin berbeda dengan ketika event dibuat. Hal ini untuk menghindari program kita untuk bertindak terlalu sensitif terhadap pergerakan mouse. (Jika kita membutuhkan lokasi yang aktual pada saat-saat tertentu, kita bisa menggunakan metode `getCursorLocation()` yang dimiliki oleh kelas `Display`.)
- Event pada mouse juga menggunakan field lain yang dinamakan `stateMask` untuk menunjukkan keadaan mouse. Seperti pada penanda tombol, `stateMask` berisi keadaan mouse sebelum terjadinya suatu event. Misalnya, jika tidak ada tombol mouse yang ditekan atau tombol keyboard lain yang ditekan ketika tombol kiri mouse ditekan, maka event mouse akan diisi dengan `button` bernilai 1 dan `stateMask` bernilai 0. `stateMask` tidak berisi "tombol 1". Akan tetapi ketika terjadi event lain ketika mouse kiri sedang ditekan, maka `stateMask` akan berisi 1.
- Keadaan suatu mouse dilambangkan oleh konstanta pada kelas SWT, seperti pada tabel berikut :

stateMask	Penjelasan
SWT.BUTTON1	Tombol 1 ditekan
SWT.BUTTON2	Tombol 2 ditekan
SWT.BUTTON3	Tombol 3 ditekan
SWT.BUTTON_MASK	Bitwise-OR dari tombol-tombol yang ditekan

- Berikut ini adalah event pada mouse yang disediakan oleh SWT. Seperti disebutkan pada bagian sebelumnya, event dan listener SWT terdiri dari event/listener tanpa tipe dan event/listener bertipe. Keduanya disarikan dalam tabel berikut :

Kelas Event (event bertipe)	**Interface/Kelas Listener** (listener bertipe)	**Metode (listener bertipe)**	**Jenis event (event tanpa tipe)**	**Penjelasan**
MouseEvent	MouseListener (dan MouseAdapter)	mouseDoubleClick(MouseEvent)	SWT.MouseDoubleClick	Mouse di-double click
		mouseDown(MouseEvent)	SWT.MouseDown	Tombol mouse ditekan
		mouseUp(MouseEvent)	SWT.MouseUp	Tombol mouse dilepaskan
MouseEvent	MouseMoveListener	mouseMove(MouseEvent)	SWT.MouseMove	Mouse berpindah posisi
MouseEvent	MouseTrackListener (dan MouseTrackAdapter)	mouseEnter(MouseEvent)	SWT.MouseEnter	Mouse masuk ke wilayah klien
		mouseExit(MouseEvent)	SWT.MouseExit	Mouse berada di sekitar klien
		mouseHover(MouseEvent)	SWT.MouseHover	Mouse keluar dari wilayah klien

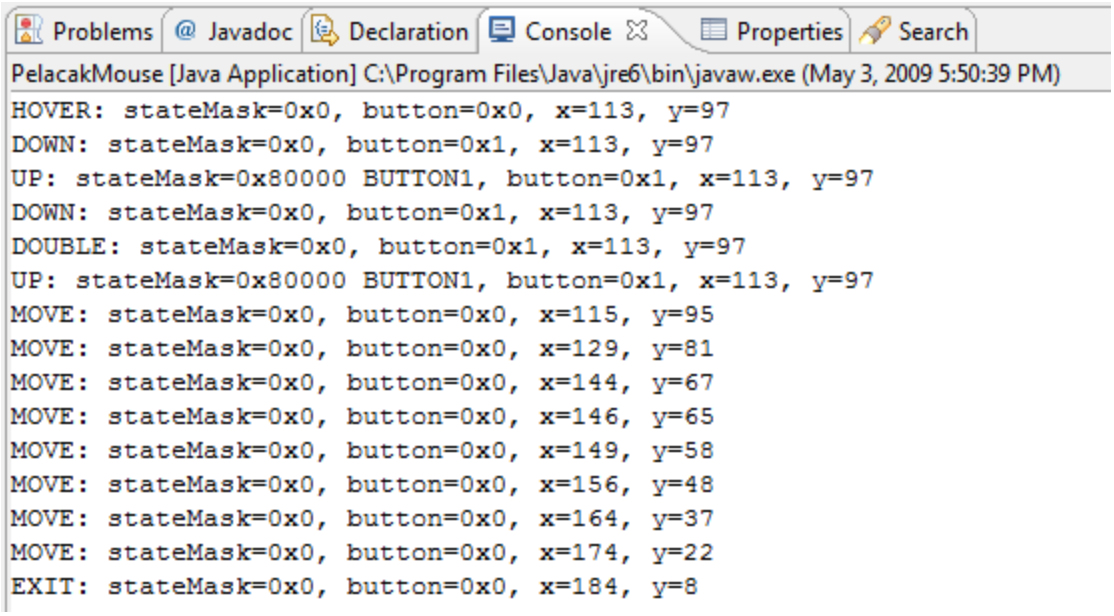
- Mari kita lihat contoh penggunaan mouse event pada program berikut. Anda bisa mengunduh program ini dan mengimportnya pada Eclipse di sini.

```
package net.lyracc.pelacakmouse;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class PelacakMouse {

    /**
     * @param args
     */
    public static void main(String[] args) {
```

```
        // Membuat display dan shell baru
        Display display = new Display();
        Shell shell = new Shell(display);

        // Kelas Listener baru, menggunakan listener tanpa tipe
        Listener mouseListener = new Listener() {

            // metode handleEvent pada interface Listener harus diimplementasikan
            public void handleEvent(Event e) {
                String output = "UNKNOWN";
                switch (e.type) {
                    case SWT.MouseDown: output = "DOWN"; break;
                    case SWT.MouseUp: output = "UP"; break;
                    case SWT.MouseMove: output = "MOVE"; break;
                    case SWT.MouseDoubleClick:
                        output = "DOUBLE";
                        break;
                    case SWT.MouseEnter: output="ENTER"; break;
                    case SWT.MouseExit: output = "EXIT"; break;
                    case SWT.MouseHover: output="HOVER"; break;
                }

                // Mengambil stateMask pada event, kemudian menampilkannya
                // dalam bentuk heksadesimal
                output += ": stateMask=0x"
                    + Integer.toHexString(e.stateMask);

                // Jika tombol Ctrl ditekan, tambahkan CTRL pada keluarannya
                if ((e.stateMask & SWT.CTRL) != 0)
                    output += " CTRL";

                // Jika tombol Alt ditekan, tambahkan ALT pada keluarannya
                if ((e.stateMask & SWT.ALT) != 0)
                    output += " ALT";

                // Jika tombol Shift ditekan, tambahkan SHIFT pada keluarannya
                if ((e.stateMask & SWT.SHIFT) != 0)
                    output += " SHIFT";

                // Jika tombol Command ditekan, tambahkan COMMAND pada keluarannya
                if ((e.stateMask & SWT.COMMAND) != 0)
                    output += " COMMAND";

                // Jika tombol kiri mouse ditekan, tambahkan BUTTON1 pada keluarannya
                if ((e.stateMask & SWT.BUTTON1) != 0)
                    output += " BUTTON1";

```

```
                // Jika tombol tengah mouse ditekan, tambahkan BUTTON2 pada keluarannya
                if ((e.stateMask & SWT.BUTTON2) != 0)
                    output += " BUTTON2";

                // Jika tombol kanan mouse ditekan, tambahkan BUTTON3 pada keluarannya
                if ((e.stateMask & SWT.BUTTON3) != 0)
                    output += " BUTTON3";

                // Mengambil field button pada event, kemudian menampilkannya
                // dalam bentuk heksadesimal
                output += ", button=0x"
                    + Integer.toHexString(e.button);

                // Mengambil koordinat x dan y
                output += ", x=" + e.x + ", y=" + e.y;

                // Menampilkan pesan keluaran pada konsol
                System.out.println(output);
            }

        };

        // Tambahkan listener pada setiap event yang ingin kita pantau
        shell.addListener(SWT.MouseDown, mouseListener);
        shell.addListener(SWT.MouseUp, mouseListener);
        shell.addListener(SWT.MouseMove, mouseListener);
        shell.addListener(SWT.MouseDoubleClick, mouseListener);
        shell.addListener(SWT.MouseEnter, mouseListener);
        shell.addListener(SWT.MouseExit, mouseListener);
        shell.addListener(SWT.MouseHover, mouseListener);

        // Ubah ukuran jendela menjadi 200 x 200 piksel
        shell.setSize(200, 200);

        // Perintah "standar" SWT, harus ada pada setiap aplikasi SWT
        shell.open();
        while (!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
    }
}
```

- Program di atas akan melacak aktivitas mouse kemudian menampilkannya pada konsol. Jangan lupa untuk menambahkan pustaka SWT sebelum menjalankan program ini seperti dibahas pada bagian ini.
- Klik Run pada Eclipse dan jalankan sebagai Java Application. Setelah jendela aplikasi yang kita buat keluar, coba jalankan mouse dan perhatikan "Console" pada Eclipse seperti gambar berikut.



Penanganan Keyboard

Membuat API yang bisa dijalankan di berbagai platform untuk mengakses keyboard bukan sesuatu yang mudah. Bahkan untuk platform yang sama tetapi dengan versi sistem operasi atau kedaerahan (locale) yang berbeda, karakter bisa dimasukkan dengan cara yang berbeda.

Metode input, atau sering juga disebut IM (input method) atau IME (input method engine), adalah mesin pengolah karakter yang disediakan oleh setiap sistem operasi. Tugasnya untuk mengubah urutan tombol keyboard menjadi karakter yang bisa dibaca sesuai dengan konfigurasi locale usernya. Misalnya, pada locale Jepang, tombol yang dimasukkan diproses oleh IME sehingga user bisa memasukkan karakter Kanji.

Untungnya, kebanyakan kontrol (widget yang bisa berinteraksi dengan user) bisa menangani keyboard tanpa perlu campur tangan aplikasi. Misalnya, ketika user memasukkan karakter Kanji pada kontrol teks, kontrol tersebut akan mengolah karakter dan menampilkannya. Jika ada menu konteks, maka menu ini akan secara otomatis disediakan.

Ketika user menekan tombol pada mouse, maka event yang dihasilkan akan diberikan kepada widget yang ada dibawahnya. Apa yang terjadi ketika tombol keyboard di tekan? Komponen mana yang harus menangani event yang dihasilkan?

Ketika suatu tombol pada keyboard ditekan, tombol tersebut akan diberikan kepada subsistem berikut dalam urutan :

1. Sistem jendela
2. Akselerator untuk menu
3. Traversal
4. Kontrol fokus

Artinya setiap subsistem diberi kesempatan untuk bereaksi terhadap tombol keyboard, dalam urutan seperti di atas. Jika salah satu subsistem mampu menangani tombol tersebut, maka tombol ini tidak akan diteruskan ke subsistem berikutnya. Dalam hal ini kita sebut tombol tersebut telah digunakan. Kita akan diskusikan setiap subsistem dari bawah ke atas, mulai dari kontrol fokus, karena konsep ini adalah konsep dasar yang diperlukan dalam pemrograman GUI.

Kontrol Fokus

Program GUI menggunakan fokus input untuk menentukan komponen mana yang yang harus menangani event dari keyboard. Pada suatu saat, hanya ada elemen pada layar yang bisa menerima input, yaitu di mana semua event dari keyboard akan diarahkan. Mungkin merupakan sesuatu yang baik untuk memberi user sedikit bantuan untuk mengetahui apakah suatu komponen memiliki fokus input. Misalnya, jika komponen tersebut adalah tempat mengetik pada program pengolah kata, maka biasanya kursornya akan berubah menjadi bentuk I dan berkedip-kedip. Contoh lainnya adalah misalnya dalam suatu formulir, input teks yang sedang dalam fokus memiliki warna latar belakang yang berbeda dengan input teks yang tidak memiliki fokus.

Untuk memberikan fokus kepada suatu komponen, kita bisa menggunakan metode `setFocus()` yang terdapat pada hampir semua widget. Jika `kontrol` adalah objek bertipe suatu widget yang bisa menerima fokus dan `sukses` adalah variabel bertipe **`boolean`**, maka

```
sukses = kontrol.setFocus();
```

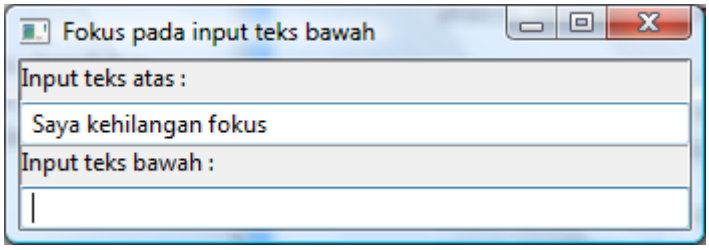
memberi perintah kepada kontrol untuk mencoba mengambil fokus input. Jika `kontrol` atau salah satu komponen yang ditampungnya berhasil mengambil fokus, fungsi ini akan mengembalikan true. Jika gagal, maka false akan dikembalikan. Widget komposit (yaitu widget yang bisa berisi widget-widget lain) akan berusaha untuk memberi fokus kepada widget yang ditampungnya sebelum mengambil fokus untuk dirinya sendiri. Beberapa widget lain, seperti label, biasanya tidak mengambil fokus. Suatu kontrol tidak bisa mengambil fokus jika ia tidak aktif atau disembunyikan, atau jika input diblokade karena modalitas (atau karena kontrol lain tidak mengijinkan fokus berpindah tempat).

Fungsi lain `kontrol.isFocusControl()` mengembalikan true jika kontrol tersebut sedang memegang fokus atau false jika tidak. Kita juga bisa mencari tahu kontrol mana yang sedang memiliki fokus dengan menggunakan metode pada kelas `Display` yaitu `Display.getFocusControl()`.

Berikut ini adalah daftar event dan listener yang berkaitan dengan fokus.

Kelas Event (event bertipe)	**Interface/Kelas Listener (listener bertipe)**	**Metode (listener bertipe)**	**Jenis event (event tanpa tipe)**	**Penjelasan**
FocusEvent	FocusListener (dan FocusAdapter)	focusGained(FocusEvent)	SWT.FocusIn	Widget menerima fokus dari keyboard
		focusLost(FocusEvent)	SWT.FocusOut	Widget kehilangan fokus dari keyboard

Mari kita lihat contoh berikut ini untuk memahami lebih lanjut tentang fokus input. Kita akan buat 2 widget yang berupa input teks. Ketika input teks atas menerima fokus, maka judul aplikasi kita ganti dengan "Fokus pada input teks atas". Ketika input teks bawah menerima fokus, maka judul aplikasi kita ganti dengan "Fokus pada input teks bawah". Jika input teks atas kehilangan fokus, kita isi input teks yang kehilangan fokus dengan "Saya kehilangan fokus", begitu input teks atas menerima fokus kembali, kita akan hapus kalimat tersebut.



Contoh program ini dapat Anda unduh di sini, untuk diimport pada Eclipse. Pembahasan detail tentang contoh program ini akan dibahas kemudian.

Event pada Tombol

Ketika suatu tombol pada keyboard ditekan, event tombol akan dibuat dan diberikan kepada aplikasi kita. Akan tetapi, tergantung pada platformnya, kedaerahan (locale), dan kombinasi tombol, ada kalanya event tidak terjadi. Misalnya, pada karakter Eropa yang memiliki aksen, mesin pengolah karakter dari sistem operasi akan mengambil tombol tersebut untuk diolah. Misalnya pada kedaerahan Jerman jika karakter ^ ditekan kemudian diikuti dengan tombol e, maka karakter ê akan ditampilkan. Demikian juga jika tombol bantu ditekan untuk mengolah bahasa Jepang, IME akan mengolah urutan karakter menjadi karakter Kanji.

Dengan kata lain, event tingkat rendah seperti ini sangat bergantung pada platform dan sistem operasi, sehingga tidak terlalu berguna untuk kebanyakan program. SWT menyembunyikan event sistem operasi dan hanya menampilkan satu event tombol saja setelah sistem operasi selesai mengolah tombol tersebut.

Berikut ini adalah daftar event dan listener yang berkaitan dengan tombol.

Kelas Event (event bertipe)	**Interface/Kelas Listener (listener bertipe)**	**Metode (listener bertipe)**	**Jenis event (event tanpa tipe)**	**Penjelasan**
KeyEvent	KeyListener (dan KeyAdapter)	keyPressed(KeyEvent)	SWT.KeyDown	Tombol ditekan
		keyReleased(keyEvent)	SWT.KeyUp	Tombol dilepaskan

Event SWT.KeyDown dan SWT.KeyUp merupakan representasi tingkat tinggi dari tombol yang ditekan dan dilepaskan. Event-event ini berguna jika kita ingin mencegat suatu tombol tertentu dan melakukan aksi khusus ketika tombol itu ditekan.

Berikut ini adalah isi event keyboard ketika ditekan/dilepaskan.

Nama Field	Penjelasan
character	Nilai Unicode dari karakter yang ditekan
keyCode	Konstanta yang menunjukkan tombol mana yang ditekan, misalnya SWT.PAGE_UP
stateMask	Melambangkan tombol tambahan, misalnya SWT.SHIFT
doit	Suatu boolean yang bisa digunakan untuk membatalkan aksi penekanan tombol

`character` berisi karakter yang kita masukkan lewat keyboard setelah diolah oleh sistem operasi. Misalnya jika kita menekan tombol <a> maka `character` berisi 'a'. Jika tombol <Shift> dan <a> ditekan, maka `character` berisi 'A'. Jika tombol <Ctrl> dan <a> ditekan, maka `character` akan diisi karakter yang bersesuaian dengan Ctrl+a, yaitu karakter dengan kode Unicode '\u0001' (atau SOH). Beberapa tombol seperti Enter, Backspace, Tab, memiliki kode karakter Unicode tersendiri. SWT juga memiliki konstanta untuk mewakili tombol-tombol ini, yaitu

Konstanta character	Penjelasan
SWT.BS	Tombol backspace
SWT.CR	Tombol Enter
SWT.DEL	Tombol Del
SWT.ESC	Tombol Esc
SWT.LF	Tombol LF
SWT.TAB	Tombol Tab

`keyCode` berisi karakter yang tidak bisa diwakilkan dengan karakter Unicode, misalnya tombol <F1>, tombol <PgUp>, tombol <Panah Atas>, dan lain-lain termasuk tombol angka pada keypad dan tombol <+> <-> <*> pada keypad. Beberapa tombol tersebut dilambangkan dalam konstanta sebagai berikut. Khusus untuk keypad, apabila tombol <+> ditekan, maka selain `keyCode` berisi SWT.KEYPADD_ADD, `character` juga berisi '+'.

SWT.F1	SWT.F11	SWT.PAGE_DOWN	SWT.KEYPAD_0	SWT.KEYPAD_EQUAL
SWT.F2	SWT.F12	SWT.HOME	SWT.KEYPAD_1	SWT.KEYPAD_CR
SWT.F3	SWT.F13	SWT.END	SWT.KEYPAD_2	SWT.HELP
SWT.F4	SWT.F14	SWT.INSERT	SWT.KEYPAD_3	SWT.CAPS_LOCK
SWT.F5	SWT.F15	SWT.KEYPAD_MULTIPLY	SWT.KEYPAD_4	SWT.NUM_LOCK
SWT.F6	SWT.ARROW_UP	SWT.KEYPAD_ADD	SWT.KEYPAD_5	SWT.SCROLL_LOCK
SWT.F7	SWT.ARROW_DOWN	SWT.KEYPAD_SUBTRACT	SWT.KEYPAD_6	SWT.PAUSE

SWT.F8	SWT.ARROW_LEFT	SWT.KEYPAD_DECIMAL	SWT.KEYPAD_7	SWT.BREAK
SWT.F9	SWT.ARROW_RIGHT	SWT.KEYPAD_DIVIDE	SWT.KEYPAD_8	SWT.PRINT_SCREEN
SWT.F10	SWT.PAGE_UP	SWT.KEYPAD_0	SWT.KEYPAD_9	

`stateMask` berisi tombol sebelum tombol ditekan, yang biasanya <Ctrl>, <Shift>, <Alt>, <Command>. Pada kebanyakan keyboard hanya ada 3 tombol pertama, akan tetapi ada juga yang memiliki lebih dari 3 tombol. Tombol-tombol ini disebut tombol pengubah.

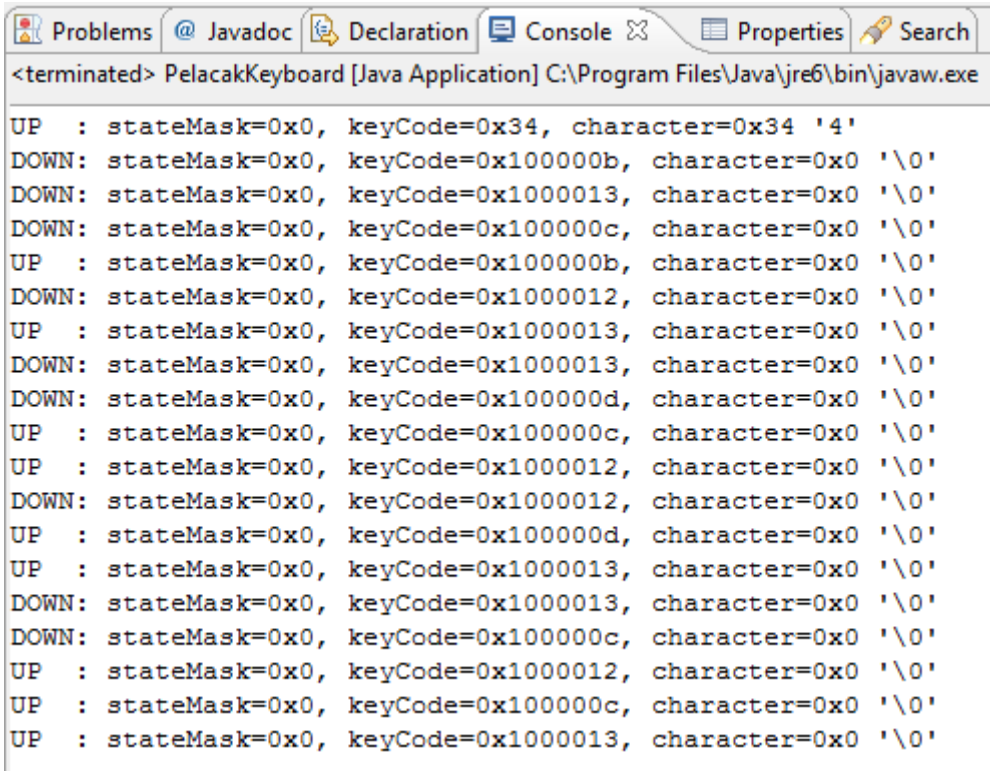
SWT membuat tombol-tombol ini menjadi kode seperti

stateMask	**Penjelasan**
SWT.MOD1	Tombol pengubah pertama ditekan (biasanya SWT.CONTROL pada Windows atau SWT.COMMAND pada Macintosh)
SWT.MOD2	Tombol pengubah kedua ditekan (biasanya SWT.SHIFT)
SWT.MOD3	Tombol pengubah ketiga ditekan (biasanya SWT.ALT)
SWT.MOD4	Tombol pengubah keempat ditekan (biasanya 0)
SWT.MODIFIER_MASK	Gabungan dari keempatnya (menggunakan bitwise OR)

Dengan representasi seperti ini, maka SWT bisa dijalankan pada beberapa platform, dan tidak bergantung dengan tombol apa yang ada pada suatu sistem operasi. Bayangkan jika Anda ingin menggunakan <Control> + <b> untuk membuat karakter menjadi tebal, akan tetapi tombol <Control> tidak tersedia pada Macintosh.

Untuk menguji tombol pengubah mana yang ditekan, kita bisa menggunakan bitwise AND, misalnya `(e.stateMask & SWT.SHIFT)`.

Berikut ini adalah contoh program pelacak keyboard yang akan melaporkan tombol apa yang Anda tekan dan lepaskan. Contoh program ini dapat Anda unduh di sini untuk mengimportnya ke dalam Eclipse. Jalankan program ini pada Eclipse, kemudian ketik apa saja di program Anda, perhatikan "Console" di Eclipse akan penuh dengan berbagai laporan tentang tombol yang ditekan dan dilepaskan.



Kadang kala dalam kondisi yang sangat langka, kita harus mengolah sendiri tombol kita sebelum diolah oleh suatu wudget. Karena SWT menggunakan widget bawaan sistem operasi, pengolahan tombol terjadi di level sistem operasi. Misalnya, ketika user mengetik pada widget teks, listener SWT.KeyDown akan dijalankan, kemudian sistem operasi akan memasukkan karakter dan menggambarnya kembali dijalankan oleh sistem operasi. Dengan menggunakan `doit`, kita bisa membuang karakter tersebut untuk tidak meneruskannya ke sistem operasi.

Contoh berikut akan menghalangi user untuk memasukkan karakter pada widget teks dengan mengeset `doit` menjadi **`false`** setiap kali event SWT.KeyDown terjadi.

```
package com.lyracc.penghalangtombol;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class PenghalangTombol {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        Text text = new Text(shell, SWT.SINGLE | SWT.BORDER);

        text.addListener(SWT.KeyDown, new Listener() {
            public void handleEvent(Event event) {
                event.doit = false;
            }
        });
```

```
        text.pack();
        shell.pack();

        shell.open();
        while (!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
    }
}
```

Traversal

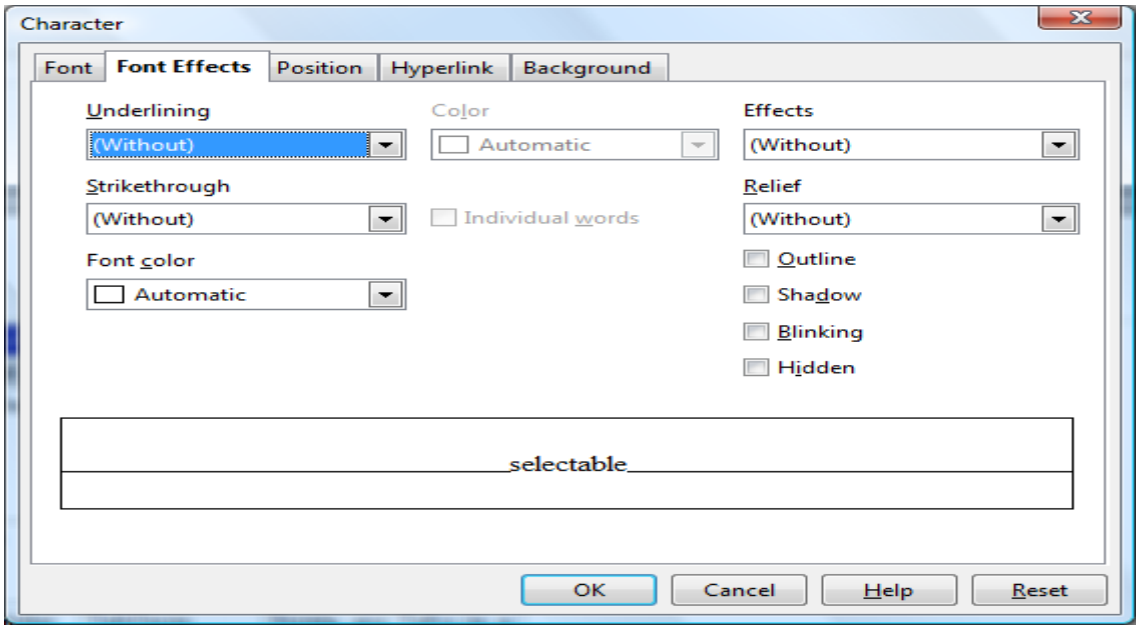
Traversal atau penelusuran berarti memindahkan fokus dari satu kontrol ke kontrol lain. Tombol traversal berbeda dengan tombol akselerasi dan tombol sistem jendela (yang akan dibahas nanti), yaitu : Suatu kontrol bebas memilih apakah akan menjalankan operasi traversal atau mengolah tombol yang ditekan. Jika kontrol memilih untuk menjalankan operasi traversal maka tombol tersebut tidak akan diproses lebih lanjut.

Ada dua jenis traversal, yaitu mnemonik dan tab.

Traversal Mnemonik

Mnemonik biasanya tertulis sebagai karakter yang diberi garis bawah pada label suatu widget. Suatu aksi akan dilakukan jika user menekan kombinasi tombol yang cocok dengan mnemonik tersebut, biasanya dengan menekan tombol <Alt> bersamaan dengan karakter yang digaris bawah tersebut.

Gambar berikut adalah contoh mnemonik pada program OpenOffice ketika kita membuka kotak dialog Format Font



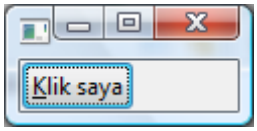
Jika kita menekan <Alt> + H maka tampilan Help akan ditampilkan.

Menekan tombol kombinasi untuk memanggil mnemonik sama dengan mengaktifkan widget tersebut. Pada contoh di atas, mnemonik <Alt> + H diterapkan pada tombol, yang artinya ketika kita menekan <Alt> + H, sama dengan kita menekan tombol Help. Beberapa widget yang tidak bisa menerima pilihan, seperti label dan kotak grup masih bisa menerima mnemonik, akan tetapi fungsinya hanya untuk memindahkan fokus ke widget tersebut.

Bagaimana caranya menambahkan mnemonik? Mudah saja. Cukup tambahkan '&' di depan karakter yang akan kita tandai sebagai mnemonik, kemudian masukkan string ini sebagai argumen pada metode `setText()` suatu widget. Misalnya pada perintah berikut :

```
Button tombol1 = new Button(shell, SWT.PUSH);
tombol1.setText("&Klik saya");
```

akan membuat tombol dengan mnemonik <Alt> + K, seperti pada gambar berikut ini :



Untuk membuat mnemonik pada karakter '&' sendiri, gunakan '&&', misalnya "Ini && Itu" akan menghasilkan "Ini & Itu"

Traversal Tab

Traversal tab didukung pada semua platform. Tidak seperti pada mnemonik, pada traversal tab, kita tidak perlu mendefinisikan apa-apa, karena setiap platform memiliki cara sendiri bagaimana memindahkan fokus dari satu widget ke widget lain. Misalnya, ketika kita menekan tombol <Tab>, maka fokus akan otomatis pindah ke widget berikutnya. Ketika sampai pada widget terakhir, maka fokus akan diulang dari widget pertama.

Nama traversal tab mungkin agak sedikit salah sasaran, karena sepertinya hanya tombol <Tab> saja yang bisa digunakan untuk memindahkan fokus dari satu widget ke widget lain. Pada beberapa platform, menekan tombol panah juga memindahkan fokus.

Beberapa tombol lain seperti <Esc> digunakan untuk menutup kotak dialog, dan sebenarnya termasuk dalam tombol traversal tab juga.

Berikut ini adalah daftar event dan listener yang berkaitan dengan traversal.

Kelas Event (event bertipe)	**Interface/Kelas Listener (listener bertipe)**	**Metode (listener bertipe)**	**Jenis event (event tanpa tipe)**	**Penjelasan**
TraverseEvent	TraverseListener	keyTraversed(TraverseEvent)	SWT.Traverse	Navigasi pada keyboard dideteksi

Berikut ini adalah isi event ketika event traversal terjadi.

Nama Field	Penjelasan
detail	Detail traversal yang terjadi
doit	Suatu boolean yang bisa digunakan untuk membatalkan aksi traversal

`detail` berisi salah satu dari nilai-nilai berikut.

isi `detail`	**Penjelasan**
SWT.TRAVERSE_ESCAPE	Traversal yang terjadi ditutupnya suatu kotak dialog, misalnya dengan menekan tombol Cancel atau tombol <Esc>
SWT.TRAVERSE_RETURN	Traversal yang terjadi ketika kotak dialog selesai diisi misalnya setelah menekan tombol OK atau tombol <Enter>
SWT.TRAVERSE_TAB_PREVIOUS	Traversal yang terjadi ketika fokus pindah ke group tab sebelumnya
SWT.TRAVERSE_TAB_NEXT	Traversal yang terjadi ketika fokus pindah ke group tab sesudahnya
SWT.TRAVERSE_ARROW_PREVIOUS	Traversal yang terjadi ketika fokus pindah ke item sebelumnya
SWT.TRAVERSE_ARROW_NEXT	Traversal yang terjadi ketika fokus pindah ke item sesudahnya
SWT.TRAVERSE_MNEMONIC	Traversal mnemonik terjadi
SWT.TRAVERSE_PAGE_PREVIOUS	Traversal yang terjadi ketika untuk pindah ke halaman sebelumnya pada kotak dialog
SWT.TRAVERSE_PAGE_NEXT	Traversal yang terjadi ketika untuk pindah ke halaman berikutnya pada kotak dialog
SWT.TRAVERSE_NONE	Traversal tidak terjadi

Variabel `detail` bukan hanya untuk dibaca akan tetapi kita juga bisa mengisinya apabila kita ingin mengubah jenis traversal. Misalnya kita ingin mengubah tombol <Enter> bukan untuk menutup dialog akan tetapi untuk memindahkan fokus ke widget lain, kita bisa mengisi variabel detail dengan `SWT.TRAVERSE_ARROW_PREVIOUS`.

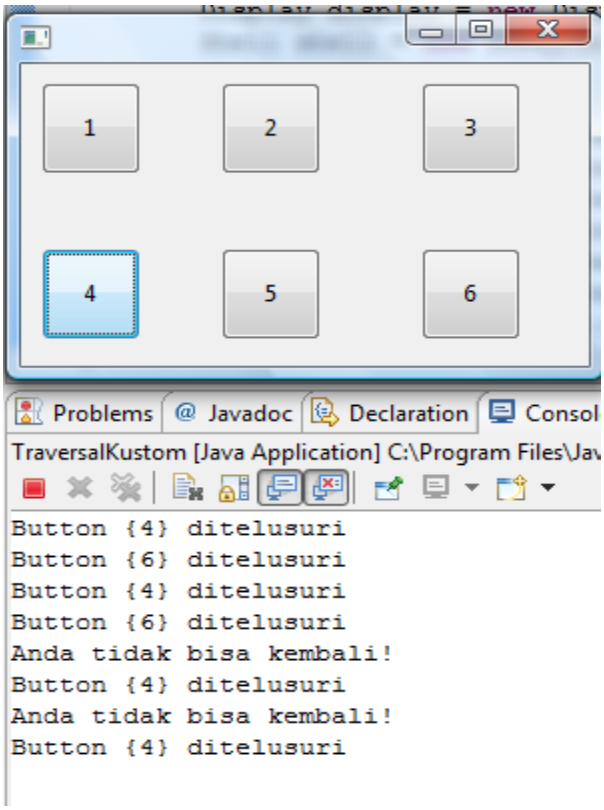
`doit` digunakan untuk membatalkan traversal jika variabel ini diisi false. Akan tetapi lihat bahwa pada variabel `detail` juga bisa `SWT.TRAVERSE_NONE`. Apa perbedaannya? Ingat bahwa tombol yang tidak digunakan untuk traversal akan diberikan kepada widget yang menerima traversal untuk diolah lebih lanjut.

Artinya jika event `doit` kita isi dengan **`true`**, traversal akan dilakukan dan tombol akan "dikonsumsi" (tidak diberikan kepada widget untuk diproses kembali). Jika `doit` kita isi denga **`false`**, traversal tidak dilakukan dan tombol akan diberikan kepada widget untuk diproses.

Apa yang terjadi jika `detail` juga diisi dengan `SWT.TRAVERSE_NONE`? Jika `detail` diisi dengan `SWT.TRAVERSE_NONE` maka widget tidak akan melakukan traversal, tidak peduli apakah isi `doit` berisi **`true`** atau **`false`**. Akan tetapi, variabel `doit` menentukan apakah tombol akan diberikan kepada widget untuk diproses.

Jadi jika `doit` berisi **`false`** dan `detail` berisi `SWT.TRAVERSE_NONE`, maka tombol akan diberikan kepada widget, dan traversal tidak dilakukan. Akan tetapi jika `doit` berisi **`true`** dan `detail` berisi `SWT.TRAVERSE_NONE`, maka traversal tidak dilakukan, dan tombol akan dikonsumsi dan tidak akan diberikan kepada widget.

Berikut ini adalah contoh penggunaan traversal.



Program ini akan membuat 6 tombol. Coba tekan tombol <Tab>. Fokus akan pindah ke tombol berikutnya setiap kali Anda menekan tombol <Tab>. Ketika Anda melewati tombol 4 atau tombol 6, akan tercetak "Button {4} ditelusuri". Jika Anda menekan tombol <Shift> + <Tab> fokus akan berpindah ke tombol sebelumnya. Akan tetapi jika Anda sampai pada tombol 4 atau tombol 6, event traversal yang terjadi akan ditangkap dan traversal akan diabaikan. Akibatnya Anda tidak akan bisa pindah dari tombol 4 ke tombol 3 atau tombol 6 ke tombol 5. Pada saat yang sama di konsol akan tercetak "Anda tidak bisa kembali!".

```
package com.lyracc.traversalkustom;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;

public class TraversalKustom {
```

```
    /**
     * @param args
     */
    public static void main(String[] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setSize(300, 200);

        Button b1 = new Button(shell, SWT.PUSH);
        Button b2 = new Button(shell, SWT.PUSH);
        Button b3 = new Button(shell, SWT.PUSH);
        Button b4 = new Button(shell, SWT.PUSH);
        Button b5 = new Button(shell, SWT.PUSH);
        Button b6 = new Button(shell, SWT.PUSH);
        b1.setBounds(10,10,50,50);
        b2.setBounds(100,10,50,50);
        b3.setBounds(200,10,50,50);
        b4.setBounds(10,100,50,50);
        b5.setBounds(100,100,50,50);
        b6.setBounds(200,100,50,50);
        b1.setText("1");
        b2.setText("2");
        b3.setText("3");
        b4.setText("4");
        b5.setText("5");
        b6.setText("6");

        TraverseListener traverseListener = new TraverseListener() {
            public void keyTraversed(TraverseEvent e) {
                if (e.detail == SWT.TRAVERSE_TAB_PREVIOUS) {
                    System.out.println("Anda tidak bisa kembali!");
                    e.doit = false;
                }
                System.out.println(e.widget + " ditelusuri");
            }
        };
        b4.addTraverseListener(traverseListener);
        b6.addTraverseListener(traverseListener);

        shell.open();
        while (!shell.isDisposed())
            if (!display.readAndDispatch())
                display.sleep();
        display.dispose();
    }
}
```

Tombol Akselerator dan Tombol Sistem Jendela

Tombol Akselerator

Akselerator adalah tombol shortcut ke salah satu menu, misalnya pada kebanyakan program pengolah kata menekan tombol <Ctrl> dan S artinya menyimpan teks yang sedang diedit. Akselerator biasanya berupa rangkaian tombol yang ditekan bersamaan, bisa 2 tombol seperti contoh <Ctrl> di atas atau mungkin lebih, misalnya pada Firefox, tombol <Ctrl> + <Shift> + J membuka konsol kesalahan (error console).

Akselerator selalu berhubungan dengan menu, sehingga akselerator bersifat global, yang berarti kita bisa menekan tombol akselerator dari dalam widget manapun yang sedang aktif pada saat itu. Ketika akselerator dipanggil, maka tombol-tombol yang ditekan akan dikonsumsi langsung oleh aplikasi dan tidak diberikan kepada widget apa-apa.

Akselerator direpresentasikan pada SWT dalam bentuk kode integer tertentu. Kode integer tersebut berisi gabungan beberapa tombol, yaitu tombol pengubah (seperti <Ctrl>, <Shift>, <Alt>) dan satu tombol lain yang berupa karakter atau keyCode (seperti tanda panah, F1 hingga F15, <Esc>, dll). Lihat bahasan tentang fokus kontrol untuk mengerti lebih jauh tentang bagaimana penanganan tombol pada SWT. Tombol akselerator tidak bisa hanya terdiri dari tombol pengubah saja.

Berikut ini adalah beberapa contoh akselerator.

Akselerator	**Rangkaian Tombol**
SWT.CONTROL + 'A'	<Ctrl> + <A>
SWT.SHIFT + SWT.ARROW_UP	<Shift> + tombol panah ke atas
SWT.MOD1 + 'S'	Tombol pengubah pertama (biasanya <Ctrl> atau <Command>) + <S>
SWT.MOD1 + SWT.MOD2 + 'B'	Tombol pengubah pertama + Tombol pengubah kedua + <B>

Akselerator diberikan kepada menu atau toolbar dengan menggunakan metode `setAccelerator(int kodeAkselerator)` di mana `kodeAkselerator` adalah kode akselerator seperti dicontohkan pada tabel di atas. Untuk mereset akselerator, isi `kodeAkselerator` dengan 0.

Untuk mengambil tombol akselerator suatu menu atau toolbar bisa digunakan dengan menggunakan metode `getAccelerator()` yang mengembalikan nilai integer.

Misalnya jika `item` adalah suatu item pada menu, maka potongan kode berikut akan menambahkan akselerator pada item tersebut :

```
item.setText("Pilih &Semua\tCtrl+S");
item.setAccelerator(SWT.MOD1 + 'S');
item.addListener(SWT.Selection, new Listener() {
    public void handleEvent(Event e) {
        System.out.println("Item dipilih.");
    }
});
```

Karakter '\t' di dalam `setText` memberi tahu SWT bahwa karakter setelah itu adalah tombol akseleratornya. Perlu dicatat bahwa hanya menambahkan teks akselerator pada metode `setText()` tidak membuat tombol akselerator otomatis ditambahkan dalam metode `setAccelerator()`. Menambahkan teks akselerator pada `setText()` berfungsi untuk membetulkan kesalahan yang terjadi jika tombol akselerator tidak terdapat pada platform yang dituju. Misalnya pada contoh di atas, tombol

<Ctrl> tidak ada pada Macintosh, akan tetapi SWT akan mengganti tampilan Ctrl-S menjadi -S.

Tombol Sistem Jendela

Tombol yang diproses oleh sistem jendela tidak akan sampai pada aplikasi kita. Misalnya, tombol Alt-F4 pada beberapa sistem akan menutup jendela yang aktid. Manager jendela akan melakukan aksi terlebih dahulu dan tombol tersebut akan dikonsumsi oleh manager jendela. Manager jendela tidak sama dengan sistem operasi. Pada Windows, manager jendela itu terintegrasi dengan sistem operasi. Pada Linux misalnya manager jendela bisa bermacam-macam, misalnya KDE dan Gnome.

Tombol-tombol mana yang akan dimanage oleh manager jendela berbeda-beda, tergantung dari sistem jendela tersebut. Karena tidak adanya kesamaan pada setiap manager jendela, maka kita tidak bisa mencegat penekenan tombol ini misalnya dialihkan untuk melakukan fungsi lain.

<!-- @page { margin: 0.79in } TD P { margin-bottom: 0in } P { margin-bottom: 0.08in } -->