

Contoh dan Penjelasan BAHASA SINGKONG Dasar-dasar Aplikasi GUI

Dilengkapi: Pengantar Rekayasa dan Pengukuran
Piranti Lunak

```
var table_orders = {}
each(keys(tables), fn(e, i) {
    set(table_orders, e, [])
})
var table_active = [0]
var table_buttons = {}

reset()
var g = component("grid", "")
var g2 = component("grid", "")
var a = component("label", "Meja")
var t = component("table", "Waktu, Kategori, Menu, Jumlah, Bungkus")
var ba = component("button", "Tambah Pesanan")
config(ba, "active", 0)
event(ba, fn() {
    var e = table_active[0]
    if (!empty(e)) {
        var ee = table_orders[e]
        var oo = add_order(e)
        if (oo != null) {
            var ee = ee + oo
            set(table_orders, e, ee)
            config(t, "contents", table_orders[e])
        }
        if (!empty(ee)) {
            set(table_status, e, 1)
            var b = table_buttons[e]
            config(b, "background", button_color[1])
        }
    }
})
grid_add(g2, a, 0, 0, 1, 1, 0, 0, 1, 1)
grid_add(g2, t, 0, 1, 1, 1, 1, 1, 3, 1)
grid_add(g2, ba, 0, 2, 1, 1, 0, 0, 1, 1)
each(keys(tables), fn(e, i) {
    var v = tables[e]
    var b = component("button", e)
    grid_add(g, b, v[0], v[1], 1, 1, 1, 1, 3, 0)
    var s = table_status[e]
    var c = button_color[s]
    config(b, "background", c)
    set(table_buttons, e, b)
    event(b, fn() {
        config(a, "text", "Meja: " + e)
        config(t, "contents", table_orders[e])
        set(table_active, 0, e)
    })
})
add([g, g2])
```



Dr. Noprianto • Dr. Karto Iskandar • Benfano Soewito, Ph.D.

Penerbit PT. Stabil Standar Sinergi

Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI

Penulis: Dr. Noprianto, Dr. Karto Iskandar, Benfano Soewito, Ph.D.

ISBN: 978-602-52770-3-0

Penerbit:

PT. Stabil Standar Sinergi

Alamat	Puri Indah Financial Tower Lantai 6, Unit 0612 Jl. Puri Lingkar Dalam Blok T8, Puri Indah, Kembangan, Jakarta Barat 11610
Website	www.singkong.dev
Email	info@singkong.dev

Cover buku:

- Masakan: singkong goreng oleh Meike Thedy, S.Kom.
- Desain cover oleh Noprianto, menggunakan GIMP. Filter-filter yang digunakan adalah: Sharpen (Unsharp Mask) dan Weave. Font yang digunakan adalah Calibri. Masing-masing teks tulisan (kecuali tulisan Penerbit) pada cover dibuat dengan beberapa layer untuk mendapatkan efek outline.
- Cuplikan kode adalah bagian dari contoh dalam buku.

Hak cipta dilindungi undang-undang.

Daftar Isi

Kata Pengantar	2
Persiapan	3
Instalasi Java dan Menjalankan Singkong.jar	5
Windows.....	5
macOS.....	8
Linux	13
Chrome OS.....	16
Tambahkan: JRE dan JDK.....	20
Tambahkan: distribusi Java alternatif	21
Contoh 1: Program GUI dalam maksimum 2 baris	23
Contoh 2: Titel, ukuran, konfirmasi tutup frame (6 baris)	25
Contoh 3: Masukkan nama dan ucapkan halo (9 atau 7 baris)	29
Contoh 4: Masukkan nama dan ucapkan halo (12 baris)	35
Contoh 5: Masukkan nama, ucapkan halo (maksimum 13 baris)	39
Rekayasa Piranti Lunak.....	45
Contoh 6: Form tambah pesanan (sekitar 50 baris).....	51
Contoh 7: Menyimpan ke file teks (+9 baris dari contoh 6)	63
Contoh 8: Simpan dan tampilkan file CSV (+10 baris dari contoh 6)	67
Contoh 9: Pesanan makanan berbeda untuk setiap meja (66 baris)	83
Contoh 10: Bekerja dengan dialog	101
Pengukuran Piranti Lunak.....	115
Bonus: Bola pantul.....	121
Daftar Pustaka	127

Kata Pengantar

Buku ini berisi sejumlah contoh source code dan penjelasan langkah demi langkah yang mudah dipahami untuk membuat program komputer yang dilengkapi Graphical User Interface (GUI), dengan bahasa pemrograman Singkong.

Anda tidak perlu memahami bahasa Singkong terlebih dahulu untuk mengikuti pembahasan dalam buku ini. Bisa menggunakan komputer dan dapat mengikuti langkah instalasi program pun sudah cukup. Pernah membuat program sebulan atau 30 tahun lalu? Itu lebih dari cukup.

Melengkapi contoh program, buku ini juga membahas pengantar untuk rekayasa dan pengukuran piranti lunak.

Jakarta, Juli 2022

Tim penulis

Buku ini membahas contoh dan penjelasan langkah demi langkah untuk program-program yang ditulis dengan bahasa pemrograman Singkong. Untuk referensi dan dokumentasi lengkap bahasa Singkong, Anda juga dapat membaca buku gratis berikut:

Mengenal dan Menggunakan Bahasa Pemrograman Singkong (ISBN: 978-602-52770-1-6, Dr. Noprianto, 2020-2022, diterbitkan oleh PT. Stabil Standar Sinergi)

Buku tersebut juga dapat di-download dari:
<https://nopri.github.io/singkong.pdf>

Persiapan

Pertama-tama, siapkanlah sebuah komputer. Boleh berbentuk laptop, desktop, ataupun server. Walaupun dengan tablet atau ponsel juga memungkinkan secara teknis, akan diperlukan pengaturan tambahan yang tidak dibahas dalam buku ini.

Untuk perangkat keras komputernya, dapat menggunakan spesifikasi komputer mulai dari yang terbaru ataupun yang telah dijual sekitar 20 tahun yang lalu. Yang penting, dapat menjalankan salah satu dari daftar sistem operasi berikut.

Interpreter Singkong (dan program yang Anda buat nantinya) dapat berjalan pada berbagai sistem operasi berikut:

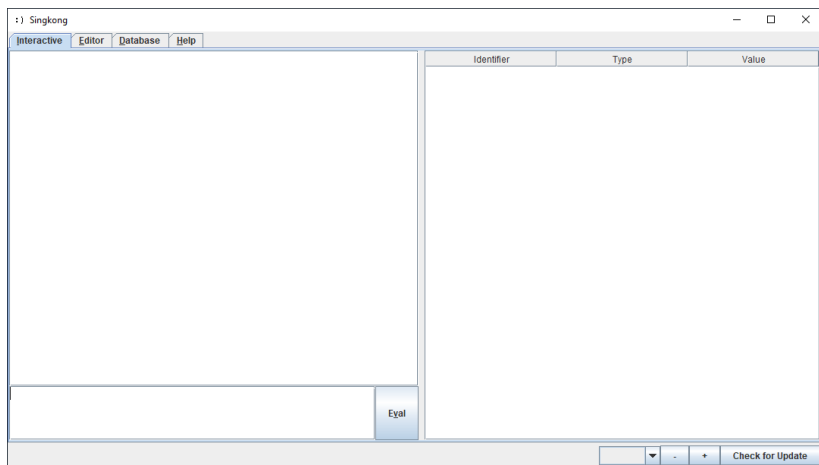
- macOS (mulai dari Mac OS X 10.4 Tiger)
- Windows (mulai dari Windows 98)
- Linux (mulai yang dirilis sejak awal 2000-an; juga termasuk Raspberry Pi OS dan Debian di Android)
- Chrome OS (sejak tersedia Linux development environment, telah diuji pada versi 101 di chromebook)
- Solaris (telah diuji pada versi 11.4)
- FreeBSD (telah diuji pada versi 13.0 dan 12.1)
- OpenBSD (telah diuji pada versi 7.0 dan 6.6)
- NetBSD (telah diuji pada versi 9.2 dan 9.0)

Setelah komputer siap, lakukanlah instalasi Java, apabila belum terinstal sebelumnya. Secara teknis, Anda hanya membutuhkan Java Runtime Environment, versi 5.0 atau lebih baru. Versi 5.0 dirilis pada tahun 2004 (sekitar 18 tahun lalu pada saat buku ini ditulis). Gunakan versi yang masih didukung secara teknis, apabila memungkinkan.

(Kenapa perlu menginstalasi Java? Karena interpreter Singkong ditulis dengan bahasa Java dan bahasa Singkong itu sendiri.)

Sebagai langkah terakhir, downloadlah interpreter Singkong, yang akan selalu didistribusikan sebagai file jar tunggal (Singkong.jar). Downloadlah selalu dari <https://nopri.github.io/Singkong.jar>. Pada saat buku ini ditulis, ukurannya hanya 4,3 MB dan berisikan semua yang diperlukan untuk mengikuti semua contoh dalam buku ini.

Apabila Singkong.jar dapat dijalankan, maka persiapan sudah selesai (silahkan langsung melanjutkan ke contoh pertama). Perhatikanlah bahwa kita akan aktif pada tab Interactive untuk menguji kode program secara langsung dan tab Editor (sebagian besar contoh) untuk mengetikkan, menyimpan/membuka, dan menjalankan kode program yang lebih panjang.



Apabila langkah detil instalasi Java dan menjalankan Singkong.jar diperlukan, bacalah juga halaman berikut.

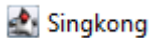
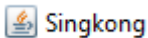
Instalasi Java dan Menjalankan Singkong.jar

Apabila diperlukan panduan teknis untuk instalasi Java, tutorial berikut membahas cara instalasi untuk sistem operasi Windows, macOS, Linux (Ubuntu), dan Chrome OS (dengan Linux development environment). Panduan ini dituliskan ulang berdasarkan konten dalam buku gratis: Mengenal dan Menggunakan Bahasa Pemrograman Singkong (ISBN: 978-602-52770-1-6, Dr. Noprianto, 2020-2022, diterbitkan oleh PT. Stabil Standar Sinergi).

Windows

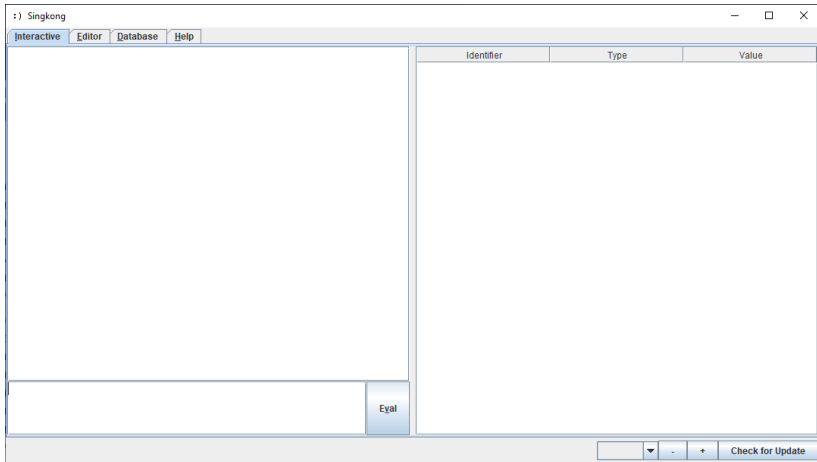
Panduan ini ditulis berdasarkan Windows 10. Sesuaikanlah apabila diperlukan.

Apabila icon Singkong.jar terlihat seperti salah satu dari gambar berikut:



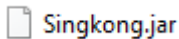
Maka, ada kemungkinan Java telah terinstal dan file .jar telah diasosiasikan untuk dibuka dengan Java.

Cobalah klik ganda pada Singkong.jar, atau klik kanan dan pilih Open. Apabila yang tampil adalah seperti gambar di halaman berikut, maka Java telah terinstal dan Singkong.jar dapat dijalankan.



Apabila Singkong.jar ternyata dibuka dengan aplikasi lain (tampilan tidak sama dengan gambar), cobalah untuk klik kanan pada Singkong.jar dan pilih Open with atau Open with.... Dari menu atau pilihan yang ditampilkan, pilihlah yang mengandung kata Java atau OpenJDK. Apabila pilihan menu tersebut tidak tersedia, maka kemungkinan besar Java belum terinstal.

Apabila icon Singkong.jar terlihat seperti gambar berikut:



Atau, ketika diklik ganda, yang tampil adalah sebagai berikut:

How do you want to open this file?



Look for an app in the Microsoft Store

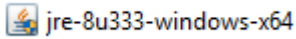
[More apps](#) ↓

Always use this app to open .jar files

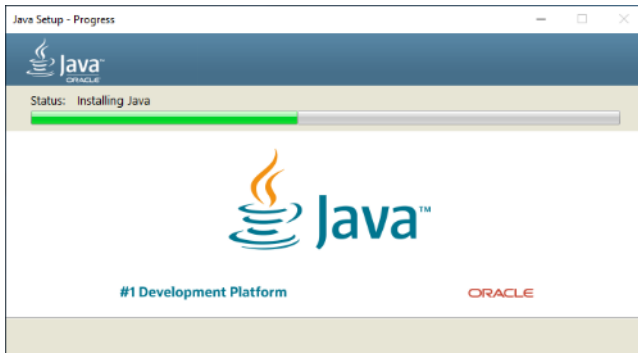
OK

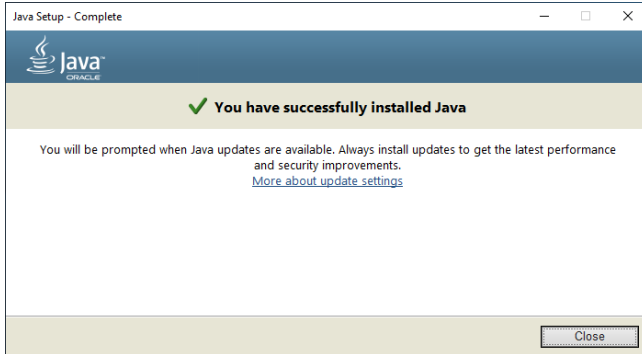
Maka, kemungkinan besar Java belum terinstal.

Untuk menginstal Java Runtime Environment, downloadlah dari java.com. Contoh file hasil download adalah seperti gambar berikut:



Jalankan file tersebut dan ikutilah langkah-langkah instalasi sebagaimana ditampilkan berikut (diawali klik pada tombol Install):





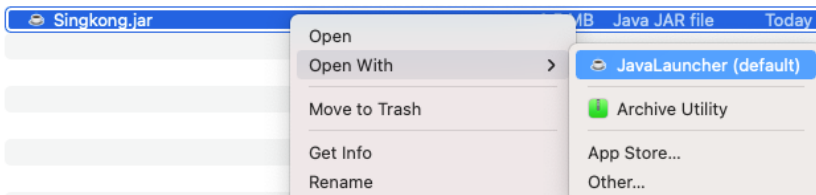
macOS

Panduan ini ditulis berdasarkan macOS 11. Sesuaikanlah apabila diperlukan.

File Singkong.jar akan tampak seperti ini pada Finder:

 Singkong.jar


Cobalah control klik (tekan dan tahan tombol control sambil klik mouse atau trackpad) pada file tersebut. Gambar serupa berikut akan ditampilkan:



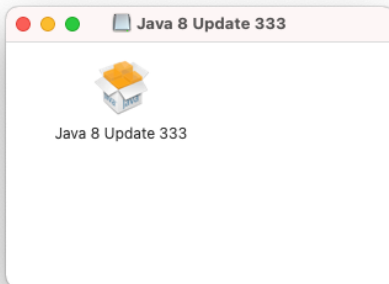
Pada menu yang tampil, pilihlah Open With, kemudian JavaLauncher (default). Apabila yang tampil adalah pesan berikut:



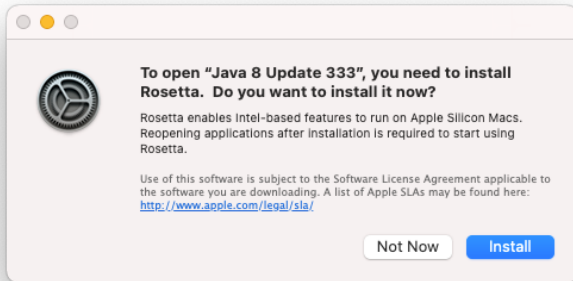
Maka, Java belum terinstal. Downloadlah dari java.com. Berikut adalah contoh file hasil download:

 `jre-8u333-macosx-x64.dmg`

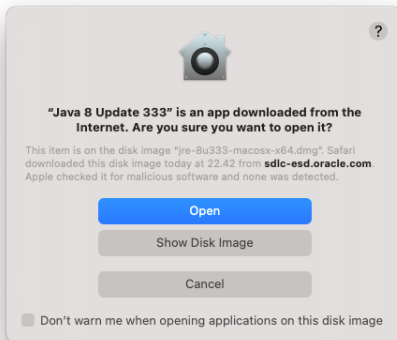
Klik gandalah untuk membuka disk image tersebut. Setelah terbuka, berikut adalah isinya:



Klik gandalah pada file installer tersebut. Apabila Mac yang digunakan menggunakan Apple Silicon, maka dialog konfirmasi untuk melakukan instalasi Rosetta mungkin akan ditampilkan:



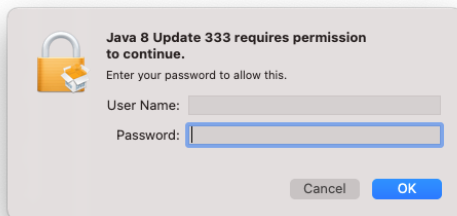
Apabila demikian, maka kliklah pada tombol Install dan tungguhlah sampai selesai. Kemudian, jalankanlah kembali file installer sebelumnya. Dialog konfirmasi serupa berikut mungkin akan ditampilkan:



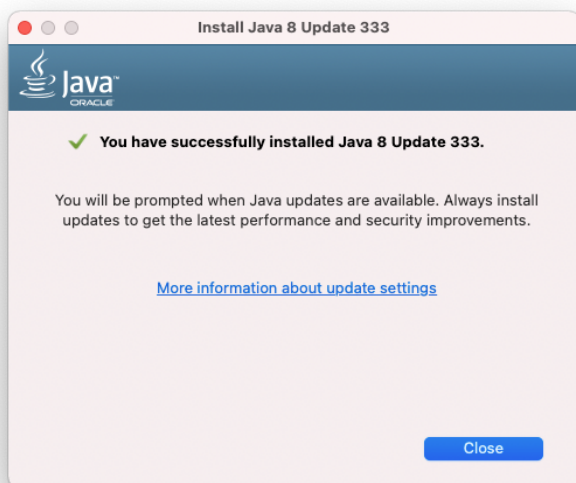
Kliklah tombol Open. Dialog instalasi berikut akan ditampilkan:



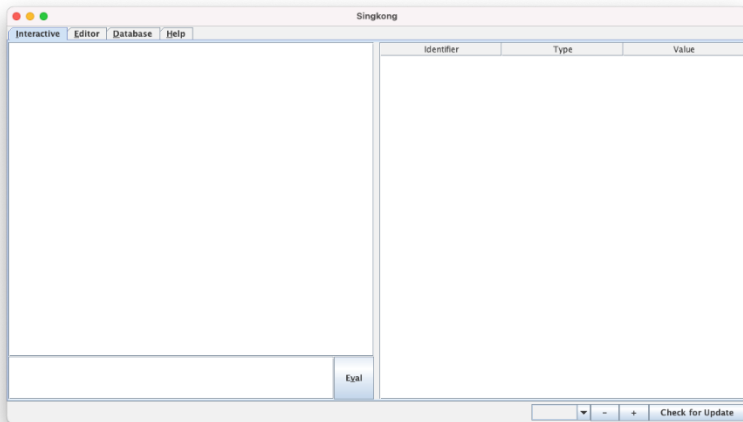
Kliklah tombol Install. Kemudian, masukkanlah password untuk pengguna yang memiliki hak instalasi software dan klik OK pada dialog yang tampil berikut:



Setelah itu, tungguilah proses instalasi Java sampai selesai.



Cobalah untuk klik ganda file Singkong.jar. Apabila yang tampil adalah seperti gambar berikut, maka Java telah terinstal dan Singkong.jar dapat dijalankan.



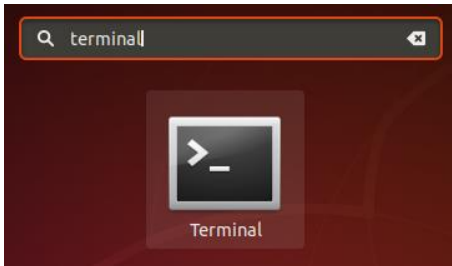
Linux

Panduan ini ditulis berdasarkan Ubuntu Linux 18.04. Sesuaikanlah apabila diperlukan.

Kliklah tombol pada bagian bawah layar seperti contoh berikut:



Kemudian, ketikkanlah terminal pada layar pencarian aplikasi berikut:



Jalankan Terminal. Kemudian, ketikkanlah java (diikuti penekanan tombol Enter) pada terminal emulator yang ditampilkan, seperti contoh berikut:

```
$ java
Command 'java' not found
```

Apabila terdapat pesan not found seperti pada gambar, maka kemungkinan besar Java belum terinstal. Masih di layar terminal emulator tersebut, apabila pengguna yang login memiliki hak sudo (apabila tidak, gunakanlah akun pengguna lain yang memilikinya), jalankanlah perintah berikut (tanpa mengetikkan \$, diikuti Enter):

```
$ sudo apt-get update
```

Masukkanlah password pengguna yang sedang login tersebut apabila diminta:

```
[sudo] password for
```

Tunggulah sampai proses update berhasil. Setelah itu, jalankanlah perintah berikut untuk instalasi Java:

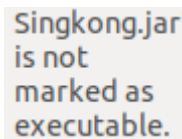
```
$ sudo apt-get install default-jre
```


Masukkanlah password pengguna yang sedang login apabila kembali diminta. Setelah itu, konfirmasilah untuk melakukan instalasi sejumlah paket yang tampil. Kemudian, tunggulah sampai selesai.

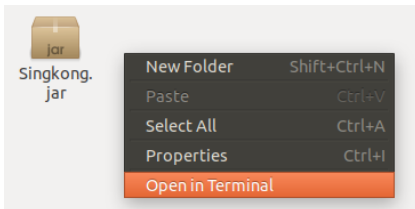
File Singkong.jar akan tampak seperti ini pada file manager:



Cobalah klik ganda pada file tersebut. Sebuah dialog dengan cuplikan pesan kesalahan berikut mungkin akan ditampilkan:



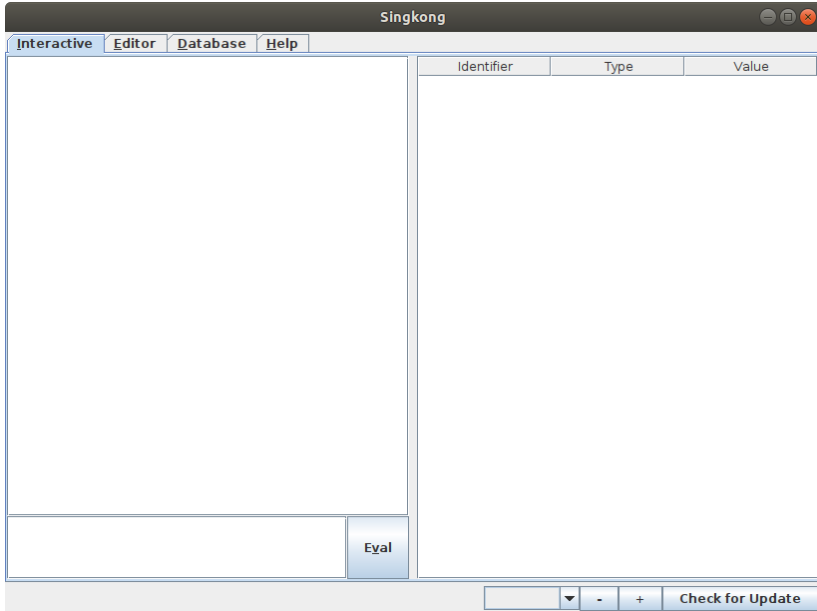
Pada area kosong daftar file di luar file Singkong.jar, klik kananlah dan jalankan Terminal, seperti contoh berikut:



Pada layar terminal emulator yang tampil, berikanlah perintah berikut:

```
$ chmod +x Singkong.jar
```

Setelah itu, kembalilah untuk klik ganda file Singkong.jar. Apabila yang tampil adalah seperti gambar berikut, maka Java telah terinstal dan Singkong.jar dapat dijalankan.



Chrome OS

Panduan ini ditulis berdasarkan Chrome OS versi 101 dengan Linux development environment telah didukung secara default. Sesuaikanlah apabila diperlukan.

Bukalah Settings (pada bagian bawah layar) dan pilihlah pengaturan Advanced seperti berikut:

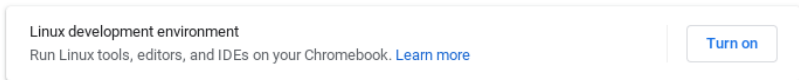
Advanced ▼

Kemudian, kliklah pada Developers:

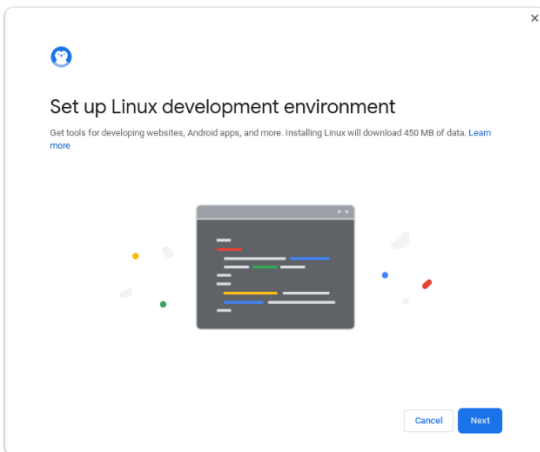
<> Developers

Pada bagian Linux development environment, kliklah tombol Turn on.

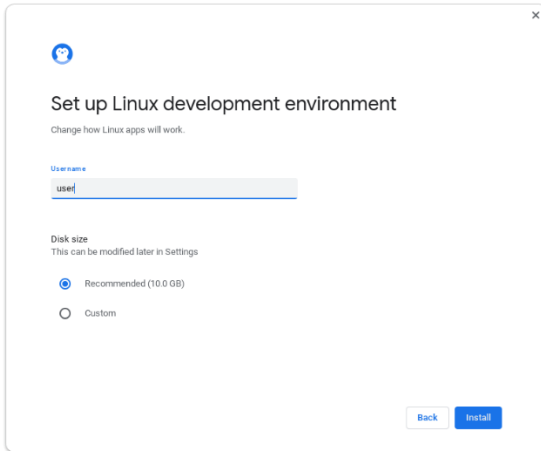
Developers



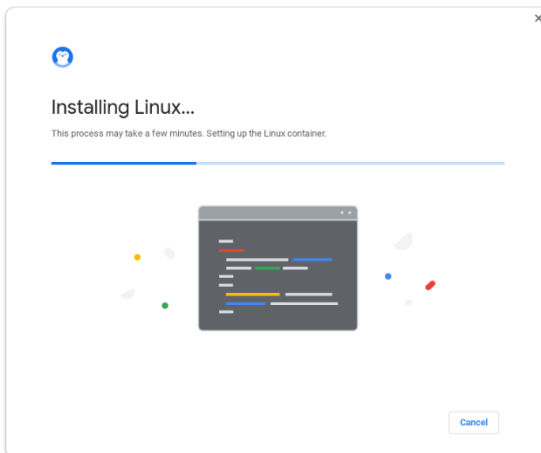
Dialog berikut akan ditampilkan. Kliklah tombol Next:



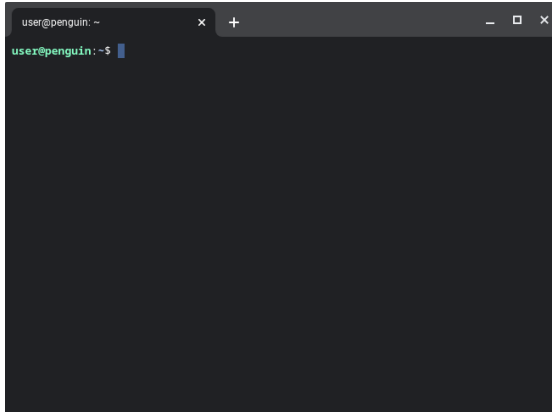
Kemudian, masukkanlah nama pengguna pada dialog yang tampil setelahnya dan kliklah tombol Install:



Setelah itu, proses download, instalasi, dan konfigurasi akan dilakukan. Tunggulah sampai selesai.



Layar terminal emulator akan ditampilkan setelah semua proses tersebut selesai. Berikut adalah contoh layarnya:



Berikanlah perintah berikut (tanpa \$, diikuti penekanan tombol Enter):

```
$ sudo apt-get update
```

Tunggulah sampai proses ini selesai. Kemudian, berikanlah perintah berikut untuk instalasi Java:

```
$ sudo apt-get install default-jre
```

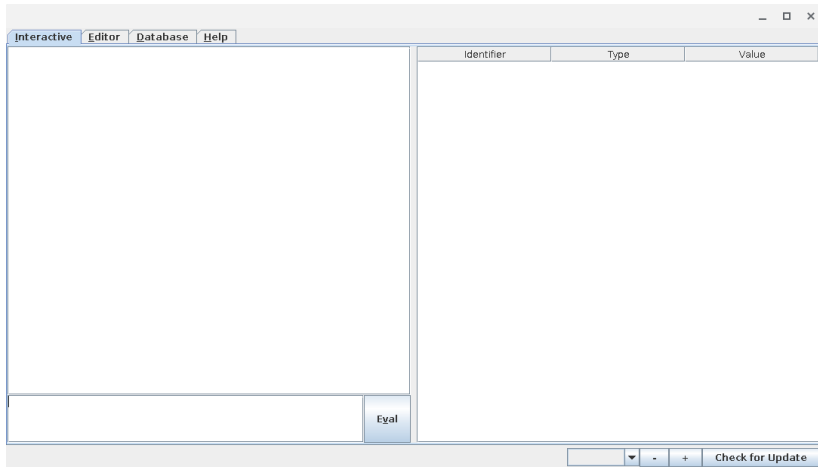
Tunggulah juga sampai selesai. Setelah itu, downloadlah Singkong.jar dengan perintah berikut:

```
$ wget https://nopri.github.io/Singkong.jar
```

Pada akhirnya, Singkong.jar dapat dijalankan dengan perintah berikut:

```
$ java -jar Singkong.jar
```

Apabila yang tampil adalah seperti gambar berikut, maka Java telah terinstal pada Linux development environment dan Singkong.jar dapat dijalankan.



Tambahan: JRE dan JDK

- Singkong.jar hanya membutuhkan Java Runtime Environment (JRE). Dengan demikian, program yang ditulis dengan bahasa pemrograman Singkong pun hanya akan membutuhkan Java Runtime Environment dan Singkong.jar.
- Dengan demikian, instalasi Java Development Kit tidak diperlukan. Java Development Kit berisikan semua yang diperlukan untuk mengembangkan dan menjalankan program yang ditulis dengan bahasa Java.
- Pembahasan instalasi Java sebelumnya mengasumsikan instalasi Java Runtime Environment. Akan tetapi, apabila instalasi Java Development Kit yang dilakukan, pun tidak akan menjadi kendala teknis, karena semua yang diperlukan untuk menjalankan Singkong.jar pun tersedia dalam instalasi tersebut.
- Instalasi Java Development Kit dapat memberikan manfaat tambahan, seperti:

- Tersedianya program `jar`, yang dapat digunakan untuk bekerja dengan arsip `.jar`. Sebagai contoh, menambahkan file ke dalam arsip `.jar`. Ini berguna, sebagai contoh, ketika program yang ditulis dengan bahasa Singkong ingin didistribusikan sebagai file `.jar` tunggal (dibundel bersama `Singkong.jar` dan semua file yang diperlukan), yang dapat dijalankan pada komputer tujuan.
 - Tanpa program `jar` tersebut, penambahan file ke arsip `.jar` tetap dapat dilakukan dengan berbagai cara lain, seperti memanfaatkan fitur yang telah disediakan oleh sistem operasi, ataupun program yang dapat menambahkan file ke format `.zip`.
- Tersedianya program `javac`, yang dapat digunakan untuk melakukan kompilasi program yang ditulis dengan bahasa Java.
 - Program yang ditulis dengan bahasa Singkong dapat memanggil method Java, yang mungkin menyediakan fungsionalitas tambahan (di luar yang telah disediakan oleh bahasa Singkong).
 - Interpreter bahasa Singkong pun dapat diembed ke dalam program yang ditulis dengan bahasa Java.

Tambahan: distribusi Java alternatif

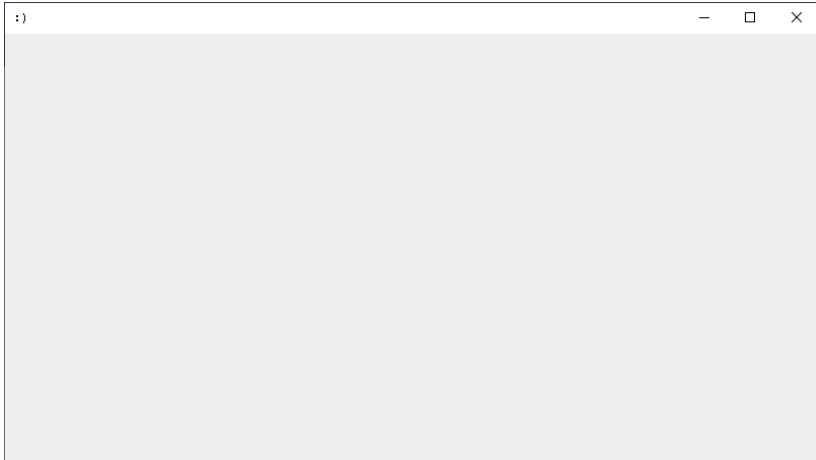
Pembahasan instalasi Java sebelumnya mengasumsikan Java Runtime Environment yang didownload dari java.com (Windows dan macOS) ataupun yang disediakan oleh distribusi sistem operasi (Linux dan Linux development environment pada Chrome OS).

Apabila untuk berbagai alasan lain (termasuk alasan non-teknis) diperlukan distribusi Java alternatif, contoh yang dapat digunakan adalah Java Runtime Environment dan/atau Java Development Kit dari adoptium.net.

Di website tersebut tersedia Eclipse Temurin, yang merupakan distribusi OpenJDK dari Adoptium.

Instalasi distribusi Java alternatif tidak dibahas dalam buku ini.

Contoh 1: Program GUI dalam maksimum 2 baris



Untuk menghasilkan program seperti ditampilkan dalam gambar, hanya diperlukan satu baris kode, yaitu:

```
show ()
```

Penjelasan:

- Fungsi show akan menampilkan frame. Di Singkong, frame adalah top level window yang tampil sebagaimana dalam gambar, dan dapat berisikan sejumlah komponen user interface seperti tombol, edit, tabel, combobox, dan lainnya.
- Show adalah fungsi built-in, atau bawaan, dan bertipe BUILTIN.
- Pemanggilan fungsi dilakukan dengan menuliskan nama fungsi, diikuti dengan tanda kurung buka (, dan kurung tutup). Fungsi dapat menerima argumen, yang akan kita bahas dalam contoh lain.
- Bahasa Singkong tidak membedakan huruf besar dan huruf kecil (tidak case-sensitive) untuk nama variabel, fungsi, dan

keyword, sehingga show() tersebut dapat dituliskan sebagai SHOW(), Show(), ataupun kombinasinya.

Lebih lanjut, judul contoh ini menyatakan dalam maksimum dua baris kode. Sementara, kita baru menulis satu baris kode, dan frame pun telah dapat ditampilkan. Anda dapat mengubah ukuran ataupun menutup frame (yang mana, program yang Anda tulis akan ditutup).

Lalu, untuk apa satu baris kode lagi?

Untuk semua contoh dalam buku ini, di awal program, kita akan menambahkan satu baris kode, yaitu:

```
reset ()
```

Sehingga, kode program keseluruhan dalam contoh ini menjadi:

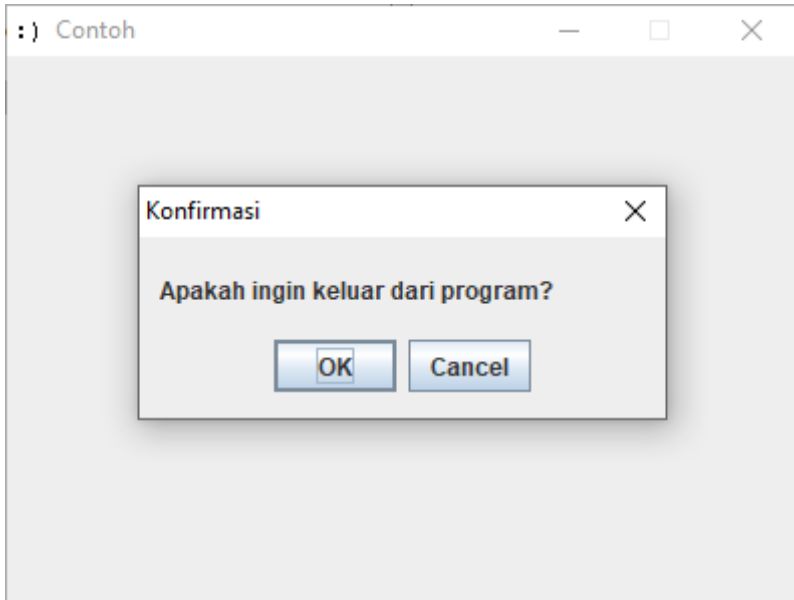
```
reset ()
```

```
show ()
```

Fungsi reset berguna untuk menghapus semua komponen dari frame, mengubah judul dan ukuran frame ke nilai default, menghentikan semua timer, meniadakan konfirmasi tutup frame, mereset status bar dan menu bar, mereset status always on top, dan menutup semua popup.

Cukup banyak guna bukan? Sebagian yang dibahas dalam penjelasan fungsi reset akan kita bahas di dalam contoh-contoh lain. Untuk sementara ini, mari kita sepakati fungsi ini akan selalu dipanggil di awal program, ketika menulis program yang berjalan di GUI (bahasa Singkong juga dapat digunakan untuk menulis program yang sepenuhnya berjalan di lingkungan kerja tanpa GUI).

Contoh 2: Titel, ukuran, konfirmasi tutup frame (6 baris)



Ukuran frame diatur dengan lebar 400 dan tinggi 300, dan secara otomatis ditempatkan di tengah layar. Perhatikanlah bahwa di bagian atas frame (titel), tertulis 'Contoh'. Lebih lanjut lagi, pada title bar, tombol maximize tidak dapat di-klik sehingga ukuran frame akan tetap sebagaimana diatur pada awalnya. Dan, pada akhirnya, ketika frame ditutup, sebuah dialog konfirmasi akan ditampilkan.

Program tersebut ditulis dalam enam baris berikut. Kita hanya akan membahas empat baris, karena reset dan show telah dibahas di contoh sebelumnya.

```
reset ()  
title ("Contoh")  
size (400, 300)
```

```
resizable(false)

closing("Apakah ingin keluar dari program?",
"Konfirmasi")

show()
```

Penjelasan:

- Fungsi title digunakan untuk mengubah title frame. Berbeda dengan contoh sebelumnya, pada contoh ini, kita memanggil fungsi dengan argumen tertentu. Dalam contoh ini, argumen yang dilewatkan adalah "Contoh", yang mana:
 - o Diapit oleh kutip dua buka " dan kutip dua tutup ". Masing-masing bukan merupakan kutip satu yang dituliskan dua kali.
 - o Ini menandakan nilai dengan tipe STRING di Singkong.
- Fungsi size digunakan untuk mengubah ukuran frame. Fungsi ini menerima dua argumen, yaitu lebar dan tinggi frame, yang diberikan sebagai NUMBER di Singkong. Perhatikanlah, karena ini merupakan NUMBER, maka tidak perlu dikutip seperti poin sebelumnya. Perhatikanlah bahwa argumen pada pemanggilan fungsi dipisahkan dengan karakter koma.
- Pemanggilan resizable dengan argumen false akan membuat ukuran frame tidak dapat diubah oleh pengguna. Dalam hal ini, false bertipe BOOLEAN di Singkong. Kebalikan dari false adalah true. Sebagaimana halnya NUMBER, tidak diperlukan kutip. Karena Singkong tidak membedakan huruf besar dan kecil, Anda dapat menuliskannya sebagai FALSE, False, ataupun kombinasinya.
- Pada akhirnya, kita memanggil fungsi closing, yang menerima dua argumen bertipe STRING. Mirip dengan contoh argumen

pada pemanggilan fungsi `title`, nilai `STRING` dituliskan dengan diapit oleh kutip dua buka “ dan kutip dua tutup “.

- Fungsi `closing` ini agak unik. Untuk argumen pertama, apabila bukan `STRING` kosong (bukan `""`), maka konfirmasi tutup frame akan diaktifkan. Apabila diberikan sebagai `STRING` kosong (`""`), maka konfirmasi ini akan ditiadakan (frame langsung ditutup tanpa konfirmasi).
- Secara otomatis, frame akan ditempatkan di tengah layar. Apabila diperlukan penempatan pada posisi tertentu, fungsi `frame_location` dapat digunakan:
 - Fungsi `frame_location` dapat menerima dua argumen opsional, yaitu posisi `x` dan `y` pada layar. Tanpa argumen, dimaksudkan sebagai tengah layar.

Sampai di sini, beberapa kali disebutkan bahwa Singkong adalah bahasa yang tidak membedakan huruf besar dan huruf kecil. Misal, `false` dapat dituliskan sebagai `False` atau `FALSE`. Atau, fungsi `show` dapat dituliskan sebagai `Show` atau `SHOW`. Akan tetapi, tentu saja ini tidak dimaksudkan untuk nilai dari sebuah `STRING`. Artinya, “Contoh” tidaklah sama dengan “contoh”. Cobalah aktif di tab `Interactive` dan ketikkanlah:

```
> "contoh" == "contoh"
true
> "contoh" == "Contoh"
false
```

Catatan: Untuk mendapatkan daftar fungsi bawaan, kita dapat:

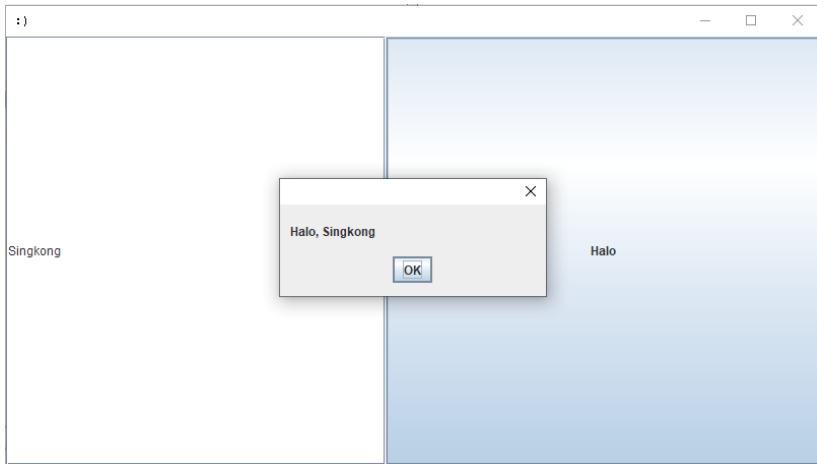
- Membacanya dari tab Help.
- Aktif di tab Interactive dan menjalankan: `builtins()`
- Membaca lebih lanjut dari buku gratis: Mengenal dan Menggunakan Bahasa Pemrograman Singkong.

Tentu saja, masing-masing dari fungsi tersebut bisa saja mengharuskan sejumlah argumen tertentu ketika dipanggil. Informasi ini, beserta penjelasan singkat fungsinya, bisa didapatkan dengan mengetikkan nama fungsi (tanpa kurung buka dan tutup) di tab Interactive, dan menekan Enter atau tombol Eval. Sebagai contoh untuk fungsi `random`:

```
> random
built-in function: random: returns random number
(between 0 inclusive and 1 exclusive), random
number between min and max (both inclusive),
random element in ARRAY, random key in HASH
arguments: 0, 1 (ARRAY or HASH), 2 (NUMBER and
NUMBER)
return value: <any type>
```

Fungsi param (mendapatkan jumlah argumen wajib) dan `help` (mendapatkan dokumentasi fungsi sebagai STRING) juga mungkin akan berguna. Anda mungkin ingin mencobanya.

Contoh 3: Masukkan nama dan ucapkan halo (9 atau 7 baris)



Astaga! Tempat memasukkan namanya (COMPONENT text) besar sekali. Begitupun dengan ukuran tombol (COMPONENT button). Walaupun program berfungsi (sebagaimana di gambar), tampilannya jelas tidak umum.

Tapi, karena program ini hanya dalam sembilan baris kode, maka cukup layak untuk dibahas, bukan? Lalu, kenapa di judul contoh kita menuliskan 'atau 7 baris'? Ini karena, dua baris hanya berupa penulisan { ataupun }).

Baiklah. Agar kode program lebih terbaca, kita tetap akan menuliskan { dan } dalam baris-baris kode tersendiri. Dengan demikian, kami berjanji tidak akan menuliskan 'atau sekian baris' lagi, hanya untuk membuat kode program lebih sulit dibaca. Jadi, tetap sembilan baris.

Dan, seperti contoh sebelumnya, kita akan kurangi fungsi reset dan show, sehingga hanya tersisa tujuh baris yang akan kita bahas.

Sebelum kita lanjut, sampai di sini, kita mengenal tipe COMPONENT, melengkapi tipe STRING, NUMBER, BOOLEAN, dan BUILTIN yang dibahas sebelumnya.

Berikut adalah keseluruhan kode programnya. Akan tampak sedikit berbeda dengan contoh-contoh sebelumnya:

```
reset()
var t = component("text", "")
var b = component("button", "Halo")
event(b, fn() {
    var m = get(t, "contents")
    message("Halo, " + m)
})
add([t, b])
show()
```

Yang pertama, munculnya kata kunci var, yang berguna untuk 'membuat' sebuah identifier/variabel. Dalam hal ini, sebuah COMPONENT text dibuat dan di-assign ke variabel t. Begitupun dengan variabel b yang merupakan sebuah COMPONENT button. Dengan demikian, kita cukup merujuk ke nama variabelnya, misal di baris kode yang diawali event dan add.

Catatan: Tentu saja, Anda mungkin ingin menggunakan nama variabel yang lebih deskriptif (diawali huruf atau underscore; diikuti huruf, underscore, atau digit), selama nama tersebut belum digunakan sebagai BUILTIN ataupun kata kunci.

Untuk membuat COMPONENT, kita menggunakan fungsi `component`. Fungsi ini menerima dua argumen wajib dan satu argumen opsional. Untuk saat ini, kita hanya menggunakan kedua argumen wajibnya:

- Tipe COMPONENT. Yaitu, salah satu dari "barchart", "button", "checkbox", "combobox", "date", "draw", "edit", "grid", "image", "label", "mask", "panel", "password", "piechart", "progress", "radio", "spin", "tab", "table", "text", "view".
- Nama COMPONENT, yang bisa berarti berbeda untuk setiap COMPONENT. Untuk saat ini, nama pada button adalah label yang ditampilkan pada button. Dengan demikian, "Halo" adalah nama untuk button, yang mana akan diartikan button akan bertuliskan: Halo.

Baris kode yang diawali dengan event tampak agak sulit terbaca. Mari kita perjelas:

```
event (b, fn () {  
    var m = get (t, "contents")  
    message ("Halo, " + m)  
})
```

Dapat kita lihat, ini sama seperti pemanggilan fungsi lain, seperti `closing`, atau `component`. Ada satu koma diantara `b` dan `fn`, sehingga fungsi `event` ini dipanggil dengan dua argumen: COMPONENT dan FUNCTION. Dan, sampai di sini, kita mengenal tipe baru lagi, yaitu FUNCTION, yang merupakan fungsi yang dibuat (bukan fungsi bawaan).

- Argumen pertama jelas, yaitu `b` (diwarnai hijau). Ini merupakan COMPONENT `button`.
- Argumen kedua yang diwarnai biru. Ini adalah sebuah FUNCTION lengkap, hanya tidak memiliki nama (seperti `show`, `reset`, dan lainnya). Karena hanya dibutuhkan pada saat pemanggilan fungsi event, kita tidak perlu repot-repot memberikan nama untuk FUNCTION tersebut (walaupun, tentu kita bisa).
- Perhatikanlah bahwa dalam hal ini, FUNCTION tidak dipanggil pada saat dilewatkan dalam fungsi event. Kita hanya membuat fungsinya. Ingatlah, untuk memanggil fungsi, kita perlu menambahkan `(`, diikuti daftar argumen fungsi apabila ada, dan `)` setelah menuliskan nama fungsinya.

Mari kita lihat lebih lanjut FUNCTION yang diwarnai biru tersebut. Ini adalah fungsi pertama yang kita buat sendiri dalam buku ini. Hore :)

```
fn() {
    var m = get(t, "contents")
    message("Halo, " + m)
}
```

Tulisan `fn` mengawali pembuatan sebuah FUNCTION. Kurung buka `(` dan kurung tutup `)` mengapit argumen yang dapat diterima oleh FUNCTION tersebut. Karena kosong, maka tidak menerima argumen.

Lalu ada `{`, yang pada akhirnya ditutup dengan `}` setelah baris `message`:

- `{` menandai awal sebuah blok, seperti halnya isi sebuah FUNCTION. Sementara, `}` menandai akhir sebuah blok. Apabila hanya dibuka dan tidak ditutup, secara sintaks bahasa Singkong, tidaklah valid (program tidak dapat berjalan).

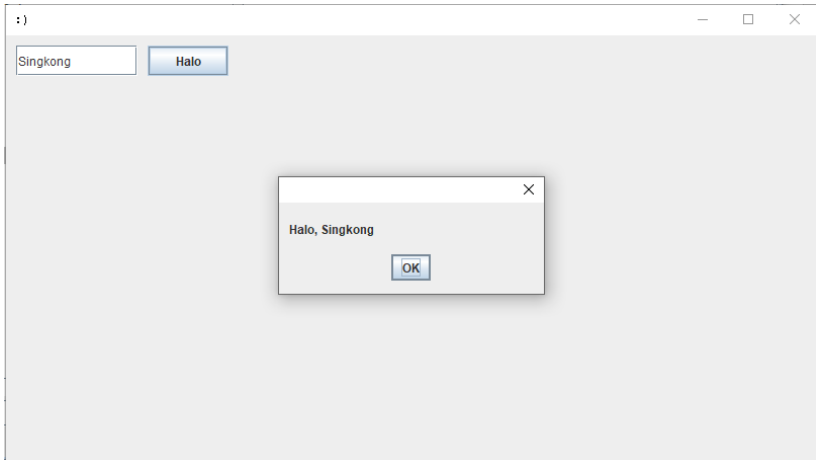
- Isi dalam fungsi (diawali var m, dan message) adalah sebagaimana baris-baris kode yang kita ketikkan pada contoh - contoh sebelumnya. Variabel yang dibuat dalam FUNCTION hanya berlaku di FUNCTION tersebut.

Di dalam FUNCTION yang dibuat, kita mendapatkan contents (isi) dari COMPONENT text (variabel t), dengan memanggil fungsi get. Kita akan membahas lebih lanjut dalam contoh-contoh lain. Untuk saat ini, untuk COMPONENT text, kita akan melewati key "contents". COMPONENT lainnya dapat berbeda.

Setelah kita mendapatkan isi dari COMPONENT text, kita tampilkan dalam sebuah message box, dengan fungsi message. Fungsi ini dapat menerima sampai dua argumen (isi dan judul), namun kita hanya melewati argumen pertama ("Halo, " digabung dengan isi text).

Halaman ini sengaja dikosongkan

Contoh 4: Masukkan nama dan ucapkan halo (12 baris)



Dibandingkan dengan contoh 3 sebelumnya, tampilan program dalam gambar kini lebih wajar. Ukuran text dan button-nya tidak berlebihan.

Kami harap Anda berkenan untuk menjalankan kembali contoh sebelumnya. Sebelum klik button Halo, cobalah untuk mengubah ukuran frame. Anda dapat melihat bahwa ukuran text dan button akan menyesuaikan dengan ukuran frame.

Sekarang, mari lihat 12 baris kode program dalam contoh ini:

```
reset ()  
var t = component ("text", "")  
var b = component ("button", "Halo")  
event (b, fn () {  
    var m = get (t, "contents")  
    message ("Halo, " + m)
```

```
}  
})  
  
var p = component("panel", "")  
panel_add(p, t, 10, 10, 120, 30)  
panel_add(p, b, 140, 10, 80, 30)  
  
add(p)  
  
show()
```

Jalankan program tersebut. Amatilah ukuran text dan button, kemudian ubahlah ukuran frame. Apa yang terjadi? Ukuran text dan button akan tetap sama, menyisakan ruang kosong di kanan dan bawah apabila frame diperbesar. Atau, button ataupun text akan terpotong apabila frame diperkecil. Apakah ini yang Anda inginkan?

Rasanya, untuk contoh ini, jawaban Anda adalah tidak.

Lalu, apakah contoh sebelumnya lebih baik?

Jawaban Anda juga mungkin tidak.

Bagaimana kalau kita atur supaya ukuran frame tidak dapat diubah? Dan, dalam contoh ini, kita pun dapat mengubah ukuran frame menjadi lebih kecil. Apakah ini adalah solusi terbaik?

Anda mungkin akan menjawab: tidak.

Lalu, kalau semua jawaban adalah tidak, apa yang dapat kita lakukan? Jawabannya adalah lanjut ke contoh 5 (walau, kami harap Anda tidak berhenti membaca buku sampai di sini saja :)).

Karena kita sudah di contoh ini, mari kita berkenalan dengan COMPONENT panel. Dengan panel, kita bisa menambahkan berbagai COMPONENT dengan posisi (x, y; dari sisi kiri atas) dan ukuran (lebar, tinggi) absolut. Dalam program tertentu (seperti yang kita bahas di bonus menjelang akhir buku), ini mungkin diperlukan.

Ketika bekerja dengan panel, kita menambahkan COMPONENT ke panel dengan fungsi `panel_add` (diwarnai biru), yang menerima 6 argumen: panel, COMPONENT tersebut, x, y, lebar, dan tinggi.

Setelah itu, barulah panel ditambahkan ke frame dengan fungsi `add` (diwarnai hijau).

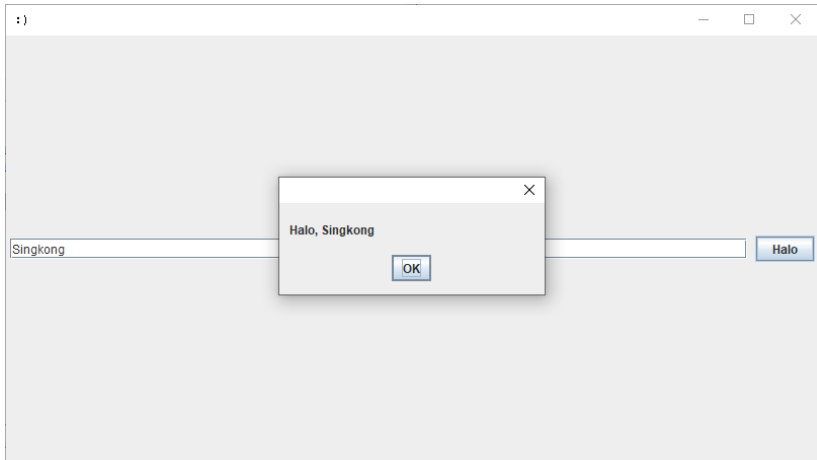
```
var p = component("panel", "")
panel_add(p, t, 10, 10, 120, 30)
panel_add(p, b, 140, 10, 80, 30)
add(p)
```

Apa dasar kita menjawab tidak, tidak, dan tidak sebelumnya? Berikut adalah beberapa alasannya:

- Ketika ukuran frame berubah, kita mungkin ingin ukuran sebagian besar COMPONENT di dalamnya ikut berubah. Ukuran button mungkin tetap akan sama. Tapi text mungkin akan menyesuaikan. Dengan demikian, idealnya tidak ada ruang kosong ataupun COMPONENT yang menjadi tidak terlihat.
- Contoh 3 sebelumnya juga tidak lebih baik. Tidaklah mungkin button sebesar itu. Anda mungkin ingin mencoba mengganti fungsi `add` di contoh 3 menjadi `add_n` atau `add_s`. Tapi, tetap tidak wajar karena ukuran text dan button akan tetap sama. Dalam contoh ini, ukuran button harusnya lebih kecil daripada ukuran text apabila ukuran frame memungkinkan.
- Mengecilkan ukuran frame dan menjadikannya tidak dapat diubah. Untuk contoh program ini? Oh, tidak! Beberapa komputer mungkin datang dengan resolusi layar yang besar. Secara umum, dengan ukuran frame tidak dapat diubah, program yang dibuat mungkin akan tampil terlalu kecil atau terlalu besar.

Halaman ini sengaja dikosongkan

Contoh 5: Masukkan nama, ucapkan halo (maksimum 13 baris)



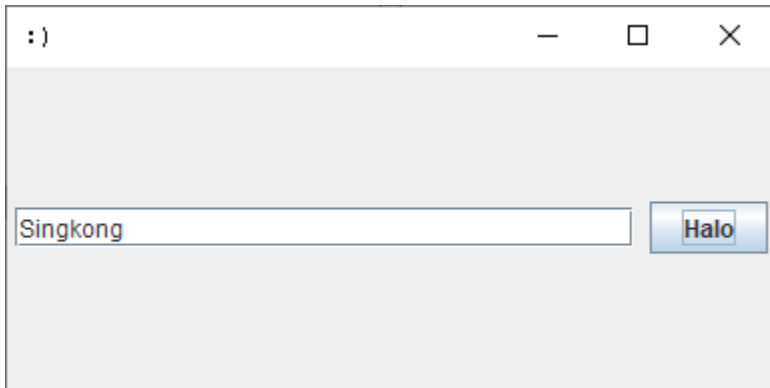
Dibandingkan dengan contoh 3 dan 4, apakah ini sudah lebih baik? Kelihatannya demikian, bukan? Mari kita lihat terlebih dahulu kode programnya:

```
reset ()  
  
var t = component ("text", "")  
var b = component ("button", "Halo")  
event (b, fn () {  
    var m = get (t, "contents")  
    message ("Halo, " + m)  
})  
  
var g = component ("grid", "")  
grid_add (g, t, 0, 0, 1, 1, 1, 1, 1, 0)  
grid_add (g, b, 1, 0, 1, 1, 0, 1, 1, 0)
```

```
#  
size(400, 200)  
;  
add(g)  
show()
```

Lho, kenapa ini 15 baris? Karena, dua baris tambahan # mengawali komentar pada source code dan baris ; mengakhirinya. Dengan demikian, pada contoh kode tersebut, baris yang diwarnai oranye tidaklah diproses oleh interpreter Singkong.

Sekarang, mari kita hapus baris # dan ;. Kemudian, jalankanlah kembali programnya:



Dibanding gambar di halaman sebelumnya, ukuran text dan button lebih wajar, karena ukuran frame tidak terlalu besar. Kita atur ukurannya dan frame secara otomatis ditempatkan di tengah layar.

Andaikata kode program yang diwarnai oranye tersebut dihapus pun, sehingga kode program menjadi kembali 12 baris, program tetap akan tampil seperti gambar pada halaman sebelumnya (dengan

asumsi, resolusi layar komputer Anda mirip dengan komputer ketika buku ini ditulis).

Lebih suka baris yang diwarnai oranye tersebut tetap ada? Jangan khawatir. Ketika pengguna program mengubah ukuran frame pun, paling jauh akan tetap tampil seperti gambar di awal contoh ini.

Sekarang, bagaimana kalau pengguna mengecilkan lebar frame sampai sesempit yang diijinkan? Mari kita lihat:



Kelihatannya masih wajar. Tidak ada COMPONENT yang tertutupi.

Kini, jawaban tidak, tidak, dan tidak pada contoh sebelumnya dapat berubah menjadi ya, ya, dan ya dengan penjelasan berikut:

- Ketika ukuran frame berubah, ukuran berbagai COMPONENT di dalamnya dapat menyesuaikan. Dalam hal ini, ukuran button akan tetap, namun text akan menyesuaikan.
- Dalam contoh ini, ukuran button tidak lagi sama persis dengan ukuran text, ketika lebar frame wajar. Bobot ukuran lebar text dalam hal ini lebih besar dari bobot lebar button, dan ketika ada ruang kosong horisontal, akan dialokasikan lebih ke text. Dalam hal ini, kedua COMPONENT akan mengisi semua ruang kosong horisontal yang teralokasikan.

- Kita memang mengatur ukuran awal frame, namun ukuran frame tetap dapat diubah oleh pengguna.

Jadi, walau esensinya kita dapat mengubah ke 12 baris kode seperti contoh 4, apa yang membuat contoh ini berbeda?

Mari kita lihat potongan kode di contoh sebelumnya:

```
var p = component("panel", "")
panel_add(p, t, 10, 10, 120, 30)
panel_add(p, b, 140, 10, 80, 30)
add(p)
```

Dan, bandingkanlah dengan potongan kode di contoh ini:

```
var g = component("grid", "")
grid_add(g, t, 0, 0, 1, 1, 1, 1, 1, 0)
grid_add(g, b, 1, 0, 1, 1, 0, 1, 1, 0)
add(g)
```

Di contoh ini, kita menggunakan COMPONENT grid, yang memungkinkan COMPONENT-COMPONENT dilayout dalam sebuah grid, dengan COMPONENT dapat:

- Menempati sel x dan y tertentu.
- Memiliki bobot lebar dan tinggi tertentu dibandingkan dengan COMPONENT lain.
- Mengisi ruang kosong yang teralokasikan, baik horisontal, vertikal, ataupun keduanya. Atau, tidak mengisi sama sekali.
- Mengatur anchor COMPONENT apakah di tengah, awal baris, akhir baris.
- Mengatur spasi antar COMPONENT yang ditambahkan (walau, kita tidak membahasnya di contoh ini).

Pada kode yang dicetak biru dengan grid, kita menambahkan text dan button:

- Text: gridx: 0, gridy: 0, gridwidth:1, gridheight: 1, weightx: 1, weighty: 1, fill: 1 (HORIZONTAL), dan anchor: 0 (CENTER).
- button: gridx: 1 (sebelah text), gridy: 0 (baris yang sama), gridwidth:1, gridheight: 1, weightx: 0 (bobot kurang dibandingkan text), weighty: 1, fill: 1 (HORIZONTAL), dan anchor: 0 (CENTER).
- Spasi antar text dan button menggunakan pengaturan default.

Sekarang, untuk contoh 3, 4, dan 5, mari kita cermati lebih lanjut:

- Fungsi add, add_e, add_n, add_s, add_w yang diantaranya digunakan pada contoh 3 dan diusulkan untuk dicoba ketika kita membahas contoh 4, akan menambahkan COMPONENT pada region center (add), east, north, south, dan west dari frame. Region center adalah yang paling besar. Masing-masing region dapat ditempatkan sejumlah COMPONENT, dengan ukuran sama. Ketika ukuran frame berubah, ukuran COMPONENT akan menyesuaikan.

north		
west	center	east
south		

- Dengan panel, kita bebas menempatkan COMPONENT di mana saja dalam panel, dengan ukuran yang diinginkan. Akan tetapi, ketika ukuran frame berubah, kita harus secara manual menyesuaikan (kalau tidak, maka akan tetap).

- Dengan grid, sebagaimana kita lihat di contoh ini, lebih nyaman dan fleksibel (misal bobot, fill, dan anchor).
- Kita bisa menambahkan COMPONENT-COMPONENT ke grid, barulah grid yang ditambahkan ke region center dengan add. Atau tambahkan ke panel, baru panelnya yang ditambahkan. Kombinasikanlah sesuai kebutuhan dan preferensi Anda.

Catatan: Sebelum melanjutkan ke contoh berikut, kita akan membahas pengantar rekayasa piranti lunak. Sehingga, pembuatan program, dalam skala tertentu, tidak hanya pada aspek penulisan source code saja.

Rekayasa Piranti Lunak

Penulis: Dr. Karto Iskandar

Rekayasa Piranti Lunak atau biasa disingkat dengan RPL secara umum disepakati sebagai terjemahan dari istilah *Software Engineering* (SE). Rekayasa Piranti Lunak adalah salah satu bidang profesi atau juga cabang ilmu pengetahuan yang mempelajari tentang pengembangan piranti lunak termasuk dalam hal pembuatan, perancangan, pemeliharaan hingga manajemen organisasi dan manajemen kualitasnya. RPL ini sendiri merupakan sebuah perubahan yang terjadi pada piranti lunak guna melakukan pengembangan, pemeliharaan, dan pembangunan kembali dengan menerapkan prinsip rekayasa (*Engineering*) sehingga memperoleh piranti lunak yang bisa bekerja secara lebih efisien dan efektif pada user nantinya.

Aplikasi atau Piranti Lunak atau *Software* adalah sebuah alat bantu terapan yang dikembangkan (*developed*) secara khusus melalui kumpulan instruksi (program komputer) untuk melakukan tugas tertentu yang dibutuhkan oleh penggunanya (*user*), yang saat dieksekusi menyediakan fitur-fitur, fungsi, dan performa yang diinginkan. Ada banyak sekali jenis Piranti Lunak, namun ada 3 jenis yang banyak dibahas saat ini, yaitu aplikasi web, aplikasi mobile, dan aplikasi desktop/komputer.

Aplikasi web, merupakan sebuah sistem yang dirancang dan disimpan pada sebuah server (biasa disebut dengan web server) kemudian dikirimkan melalui jaringan internet dan diakses oleh pengguna melalui browser dari seluruh dunia yang terhubung dengan jaringan internet. Dalam mengatur komunikasi dan integrasi jaringan, digunakan standar protokol internet yaitu TCP/IP. TCP memiliki tugas untuk memastikan bahwa data yang dikirim dapat terkirim dengan sukses, sedangkan IP adalah alamat komputer dari suatu jaringan bertugas untuk mentransmisikan paket data dari satu

komputer ke komputer lainnya agar paket yang di kirim dapat diterima di tujuan yang tepat.

Aplikasi mobile, merupakan sebuah sistem yang dirancang khusus untuk platform mobile dengan segala fitur dan fasilitas pendukung di dalamnya. Saat ini terdapat 2 jenis platform mobile yang umum digunakan, yaitu iOS dan Android. Peningkatan pesat dalam penggunaan aplikasi berbasis mobile sangat menginspirasi para pengembang untuk membuat aplikasi yang realtime dan tidak biasa. Banyak fitur dan fasilitas dari perangkat smartphone atau gadget yang dapat memperkaya aplikasi mobile, seperti notification, Global Positioning System (GPS), Geolocation, layar sentuh, fitur kamera, Short Message Service (SMS), Google Maps API, local data access, dan masih banyak lagi.

Aplikasi desktop/komputer, merupakan sebuah sistem yang dirancang khusus untuk sebuah Operating System (OS) tertentu, misalnya Windows, Mac, Linux, ataupun OS lainnya. Aplikasi jenis ini harus terinstall pada OS tersebut dan issue kompatibilitas perlu diperhatikan terkait versi dari OS nya. Banyak tools ataupun bahasa pemrograman yang mendukung pengembangan aplikasi jenis ini, termasuk Bahasa pemrograman Singkong yang dibahas pada buku ini.

Banyak orang khususnya programmer beranggapan membuat aplikasi yang penting bisa coding, sudah cukup. Semakin pandai coding dan tanpa error dianggap sebagai modal yang cukup untuk mengembangkan aplikasi. Apakah betul begitu?

Tentu saja tidak, terutama ketika kita ingin membangun sistem aplikasi yang kompleks dan semakin kompleks, maka diperlukan manajemen didalamnya agar efektif serta efisien dalam penggunaan waktu yang panjang dan tentu saja harus memperhatikan sistem aplikasi yang dihasilkan haruslah memiliki kualitas yang baik.

Oleh karena itu, tujuan mempelajari rekayasa piranti lunak ini adalah dengan biaya produksi dan perawatan piranti lunak yang lebih rendah, menghasilkan piranti lunak yang mampu bekerja dengan baik, serta mampu menghasilkan piranti lunak yang kinerjanya handal dan tepat waktu.

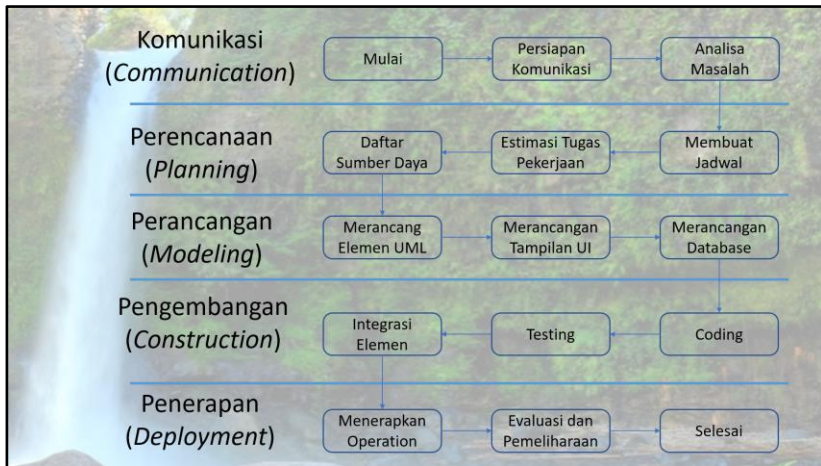
Kriteria piranti lunak yang harus diperhatikan dalam proses rekayasanya adalah sebagai berikut:

1. Piranti lunak haruslah dapat terus dirawat dan dipelihara (*maintainability*)
2. Piranti lunak haruslah dapat mengikuti perkembangan teknologi (*dependability*)
3. Piranti lunak haruslah dapat mengikuti keinginan pengguna (*robust*)
4. Piranti lunak haruslah efektif dan efisien dalam menggunakan energi dan penggunaannya
5. Piranti lunak haruslah dapat memenuhi kebutuhan yang diinginkan (*usability*)

Melihat pentingnya RPL dalam pengembangan piranti lunak, maka banyak model pengembangan piranti lunak yang berkembang dan dapat dipergunakan. Pada buku ini akan dibahas satu model pengembangan piranti lunak yang paling klasik, yaitu Model Sekuensial Linier atau *Model Waterfall*.

Model Waterfall mendefinisikan beberapa fase/tahapan berurutan yang harus diselesaikan dan melanjutkan ke fase/tahapan selanjutnya membutuhkan fase/tahapan sebelumnya selesai secara keseluruhan. Oleh karena itu *Model Waterfall* sering dikatakan bersifat sistematis dan berurutan dalam membangun sebuah piranti lunak. Model Waterfall terdiri dari 5 fase/tahapan untuk pengembangan, yaitu: Komunikasi (*Communication*), Perencanaan (*Planning*), Perancangan (*Modeling*), Pengembangan (*Construction*),

dan Penerapan (*Deployment*). Sebagai ilustrasi *Model Waterfall* dapat dilihat pada gambar berikut:



Sesuai nama *Model Waterfall*, cara bekerja model ini menyerupai sebuah air terjun. Model pengembangan piranti lunak ini cocok untuk pengembangan piranti lunak untuk sebuah kasus atau permasalahan yang sudah jelas dan tidak terlalu kompleks. Berikut penjelasan detail masing-masing fase/tahapan:

1. *Komunikasi (Communication), Project Initiation & Requirements Gathering*

Sebelum memulai pekerjaan yang bersifat teknis, sangat diperlukan adanya komunikasi dengan pengguna (*user*) demi memahami dan mencapai tujuan yang ingin dicapai. Hasil dari komunikasi tersebut adalah inisialisasi proyek seperti menganalisis permasalahan yang dihadapi dan mengumpulkan data-data yang diperlukan, serta membantu mendefinisikan fitur dan fungsi dari piranti lunak. Komitmen awal menjadi kunci penting dalam keberhasilan pengembangan piranti lunak secara keseluruhan. Fase/tahapan ini sangat penting dan haruslah

diselesaikan terlebih dahulu sebelum melanjutkan ke fase/tahapan selanjutnya.

2. Perencanaan (*Planning*), *Estimating*, *Scheduling*, *Tracking*

Tahap berikutnya adalah tahapan perencanaan yang berisikan membuat jadwal, menjelaskan tentang estimasi tugas-tugas teknis yang akan dilakukan, resiko-resiko yang dapat terjadi, sumber daya yang diperlukan dalam membuat sistem, produk kerja yang ingin dihasilkan, detail penjadwalan kerja yang akan dilaksanakan dan tracking proses pengerjaan sistem. Fase/tahapan perencanaan ini harus diselesaikan baru kemudian dilanjutkan ke fase/tahapan perancangan berikutnya.

3. Perancangan (*Modeling*), *Analysis & Design*

Tahapan ini adalah tahap perancangan dan pemodelan arsitektur sistem yang berfokus pada perancangan infrastruktur, elemen UML, struktur data, arsitektur software, tampilan interface, dan algoritma program. Tujuannya untuk lebih memahami gambaran besar dari apa yang akan dikerjakan. Hasil dari fase/tahapan perancangan ini penting untuk dikomunikasikan ke pengguna dan perlu mendapatkan persetujuan untuk dibawa ke Fase/tahapan pengembangan berikutnya. Perubahan pada hasil fase/tahapan ini lebih mudah dilakukan daripada mengubah piranti lunak nantinya.

4. Pengembangan (*Construction*), *Code & Test*

Tahapan ini merupakan proses penerjemahan bentuk perancangan desain menjadi kode atau bentuk bahasa yang dapat dibaca oleh mesin. Dalam buku ini menggunakan bahasa pemrograman Singkong. Setelah pengkodean selesai, dilakukan pengujian terhadap sistem dan juga kode yang sudah dibuat. Untuk aplikasi kompleks mungkin piranti lunak terdiri dari

beberapa elemen yang perlu digabungkan menjadi satu. Perlu dilakukan pengujian masing-masing elemen setelah penggabungan tersebut. Tujuannya untuk menemukan kesalahan yang mungkin terjadi untuk nantinya diperbaiki. Fase/tahapan pengembangan ini harus diselesaikan baru kemudian dilanjutkan ke fase/tahapan penerapan berikutnya.

5. Penerapan (*Deployment*), *Delivery*, *Support*, & *Feedback*

Tahapan terakhir ini merupakan tahapan implementasi piranti lunak ke pengguna (*user*), perbaikan piranti lunak, evaluasi piranti lunak, dan pengembangan piranti lunak kembali berdasarkan umpan balik yang diberikan agar sistem dapat tetap berjalan dan berkembang sesuai dengan fungsinya. Fase/tahapan akan dilanjutkan kembali ke fase/tahapan awal pengembangan piranti lunak dengan memulai siklus baru.

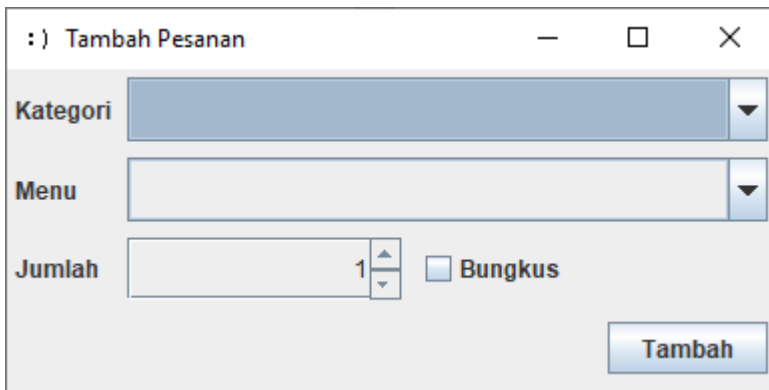
Soal Latihan:

1. Coba cari beberapa model pengembangan piranti lunak lain yang anda kenal! Dan bandingkan dengan *Model Waterfall* yang telah dibahas dalam buku ini.
2. Analisa masing-masing fase/tahapan pada proses pengembangan piranti lunak *Model Waterfall*, dan terapkan pada pengembangan piranti lunak yang pernah anda kerjakan, ubah-ubah masing2 detail fase/tahapan disesuaikan dengan piranti lunak yang anda kembangkan.
3. Menurut anda, apakah ada perbedaan detail fase/tahapan *Model Waterfall* untuk pengembangan aplikasi web, aplikasi mobile, dan aplikasi desktop/komputer?
4. Coba analisa, apakah ada perbedaan pada *Model Waterfall* jika piranti lunak yang akan dikembangkan menggunakan bahasa pemrograman Singkong?

Contoh 6: Form tambah pesanan (sekitar 50 baris)

Apa? Dari maksimum 13 baris kode, langsung ke sekitar 50 baris? Ini karena, kami yakin Anda dapat mengikuti contoh ini dengan mudah. Semua yang terkait penempatan COMPONENT pada frame telah dicontohkan sebelumnya. Memang ada COMPONENT baru, tapi tidaklah sulit.

Mari kita lihat formnya ketika ukuran belum diubah:

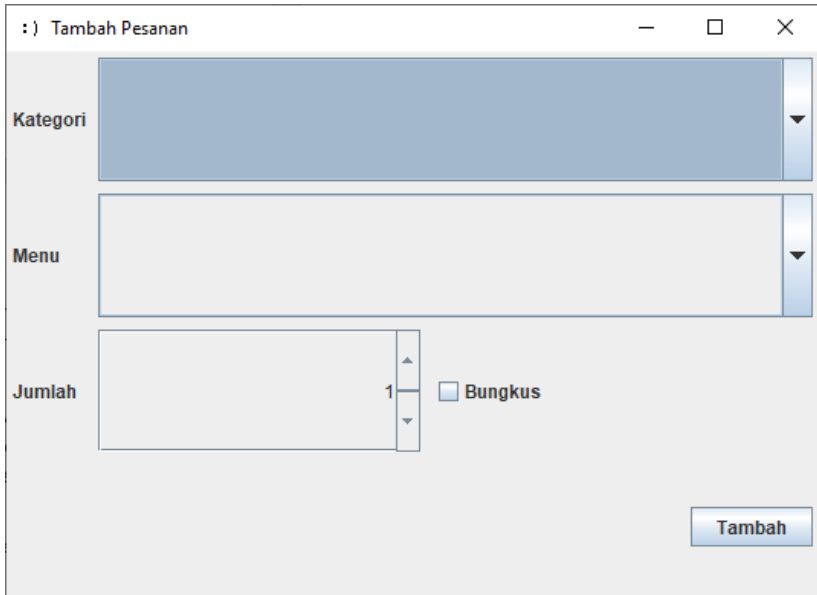


Bisa dilihat, ukuran combobox (untuk Kategori dan Menu) dan spin (Jumlah) tampak sedikit lebih tinggi (mengisi ruang teralokasi) dibandingkan dengan checkbox (Bungkus), button (Tambah), dan label (tulisan Kategori, Menu, dan Jumlah).

Lalu, ketika pengguna mengubah ukuran frame menjadi lebih tinggi dan lebar:

- Akan tersedia ruang kosong tambahan.
- COMPONENT yang ada dapat diatur agar, sesuai bobotnya, menempati ruang kosong yang ada ataupun tetap.

Berikut adalah gambar setelah frame diubah ukurannya:



Perhatikanlah bahwa combobox Kategori, Menu, dan spin Jumlah ikut meninggi dan melebar juga. Akan tetapi, ukuran checkbox Bungkus dan button Tambah, tetap.

Mari kita lihat cuplikan kode pembuatan COMPONENT untuk combobox dan spin berikut. Untuk label dan checkbox, kita tidak membahasnya karena serupa dengan button yang telah dibahas pada contoh sebelumnya.

```
var c1 = component("combobox", ",Makanan,Minuman")
var c2 = component("combobox", "")
var s1 = component("spin", "1,1,20,1")
```

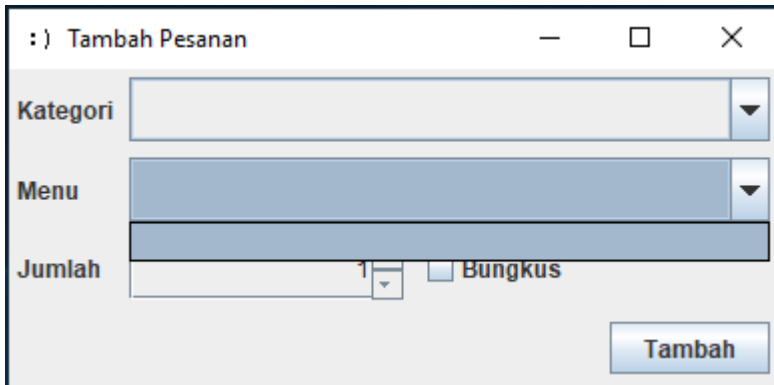
Sebagaimana disebutkan pada contoh 3, nama COMPONENT (argumen kedua pada fungsi component) bisa berarti berbeda, sesuai dengan COMPONENT yang dibuat. Sebagian ditampilkan pada COMPONENT itu sendiri, namun:

- Pada combobox: pilihan yang ada. Pada c1, terdapat 3 pilihan: yaitu (kosong), Makanan, Minuman. Untuk item pertama, memang sengaja kita kosongkan.
- Pada spin: nilai aktif, minimum, maksimum, langkah. Pada s1, ini berarti secara default nilai yang tampil adalah 1, dengan rentang pilihan 1 sampai 20, dan setiap penambahan adalah 1 (misal setelah 3 adalah 4).

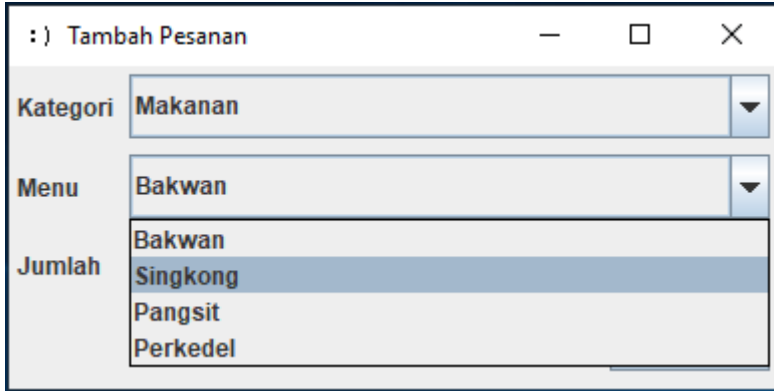
Kita mengubah sedikit contoh pemanggilan `grid_add` seperti dibahas pada contoh 5 untuk c1 dan b1 (button Tambah):

- Combobox c1: `gridx: 1` (sebelah label Kategori), `gridy: 0` (baris pertama), `gridwidth:2`, `gridheight: 1`, `weightx: 1` (bobot lebih dibandingkan label), `weighty: 1`, `fill: 3` (BOTH; HORIZONTAL dan VERTICAL), dan `anchor: 1` (LINE_START).
- Button b1: `gridx: 1` (2, kolom paling kanan dalam form ini), `gridy: 3` (baris keempat), `gridwidth:1`, `gridheight: 1`, `weightx: 1`, `weighty: 1`, `fill: 0` (NONE), dan `anchor: 2` (LINE_END).

Sekarang, mari kita lihat bahwa isi combobox Menu bergantung pada apa yang dipilih pada combobox Kategori:



Pada gambar di atas, combobox Menu kosong karena Kategori belum dipilih. Mari kita pilih Kategori tertentu:



Pada gambar di atas, ketika Makanan dipilih dari Kategori, pilihan combobox Menu pun tersedia.

Mari kita lihat, untuk contoh ini, apa yang menjadi dasar ketersediaan Menu:

```
var products = {  
    "makanan": ["Bakwan", "Singkong", "Pangsit",  
    "Perkedel"],  
    "minuman": ["Air", "Kopi", "Teh"]  
}
```

Sebuah variabel dengan nama products:

- Bertipe HASH, pemetaan key ke value, dengan key dan value dapat bertipe apa saja. Urutan pemetaan ditambahkan akan diingat (misal pemetaan mana yang duluan dibuat). Jumlah pemetaan tidak dibatasi.
- Terdapat dua key, masing-masing bertipe STRING dengan isi semua adalah huruf kecil: "makanan", "minuman".
- Masing-masing key dipetakan ke nilai dengan tipe berupa ARRAY. Sekilas, kita sudah melihat penggunaan ARRAY ketika menggunakan fungsi add di contoh 3.

- Masing-masing isi ARRAY bertipe STRING. Akan tetapi, ARRAY di Singkong dapat berisikan tipe apa saja, termasuk ARRAY (dan dapat bersarang; ARRAY dalam ARRAY dan seterusnya). Urutan sesuai indeks akan diingat dan jumlah elemen tidak dibatasi.

Perhatikanlah kembali baris kode berikut:

```
var c1 = component("combobox", "Makanan,Minuman")
```

Kini, kita dapat melihat kaitan antara apa yang pengguna pilih dari combobox c1. Apabila "Makanan", kita tinggal pastikan diubah ke huruf kecil semua ("makanan"), karena key products semua dalam huruf kecil.

Sekarang, mari kita perhatikan, apa saja tindakan dari pengguna yang harus kita proses:

- Ketika pengguna mengubah pilihan pada combobox Kategori: combobox Menu akan diisi dengan pilihan yang sesuai.
- Ketika pengguna menekan button Tambah: untuk saat ini, sebuah message akan ditampilkan.

Perhatikanlah gambar-gambar di halaman berikut, ketika kita:

- Memilih Kategori: Makanan
- Memilih Menu: Singkong
- Menentukan Jumlah: 6
- Mencentang Bungkus

Pada saat button Tambah di klik:

Untuk contoh ini, kita tidak menggunakan database untuk mengisi combobox Kategori, Menu, ataupun ketika button Tambah diklik. Yang penting, event ketika pilihan combobox berubah dan button diklik harus diproses terlebih dahulu.

Catatan: Bahasa pemrograman Singkong mendukung pemrograman yang terhubung ke sistem database relasional dan datang dengan driver, fungsi-fungsi bawaan, dan modul bawaan untuk kebutuhan tersebut. Akan kita bahas pada buku lain, yang juga dapat didownload dengan gratis.

Sejumlah (tapi tidak semua) COMPONENT di Singkong dapat menerima event default. Untuk combobox, ini berarti pilihan diubah. Dan, untuk button, ini berarti ketika ditekan. Event default COMPONENT lain dan event-event lain akan kita bahas pada contoh-contoh dan buku-buku lainnya.

Sebagaimana kita lihat di contoh 3, untuk memproses event default, kita menggunakan fungsi event. Mari kita lihat ketika pilihan combobox Kategori diubah:

```
event(c1, fn() {  
    var c = lower(get(c1, "text"))  
    var p = products[c]  
    if (p == null) {  
        var p = []  
    }  
    config(c2, "contents", p)  
})
```

Penjelasan:

- **Diwarnai merah:** Apabila kita mendapatkan “contents” pada contoh 3, untuk mendapatkan item yang terpilih pada sebuah combobox, kita mendapatkan “text” ketika memanggil fungsi get. Kemudian, kita ubah ke huruf kecil semua dengan fungsi lower.
- **Diwarnai biru:** untuk mendapatkan value pemetaan dari key di HASH, kita menggunakan [dan], dengan nama key diapit oleh [dan] tersebut. Dengan demikian, pada baris ini, ada kemungkinan kita mendapatkan pilihan berupa: “”, “makanan”, ataupun “minuman”. Perhatikanlah bahwa pada

saat combobox dibuat, kita menyediakan sebuah pilihan "". Dan kita sudah ubah semua ke huruf kecil.

- **Diwarnai hijau:** ketika value tidak ditemukan pada HASH untuk key tertentu, null akan dikembalikan. Nilai null bertipe NULL. Dalam hal ini, apabila null yang didapatkan (berarti, pengguna memilih pilihan "", yang tidak tersedia pemetaan di HASH products), maka sebuah ARRAY kosong ([]) diassign ke variabel p.
 - o Kita menggunakan if memeriksa kondisi/melakukan seleksi dan operator == (mengetikkan simbol sama dengan dua kali) dapat digunakan untuk membandingkan apakah kedua operan sama.
- **Diwarnai oranye:** untuk menentukan daftar item yang dapat dipilih untuk sebuah combobox, kita akan:
 - o Menggunakan fungsi config. Fungsi ini dapat menerima 4 argumen. Akan tetapi, untuk contoh ini, kita hanya akan menggunakan 3 argumen saja.
 - o Untuk combobox, "contents" berarti daftar item.
 - o Dan, untuk combobox dan "contents", sebuah ARRAY digunakan.
 - Entah kita tentukan [] seperti yang diwarnai hijau,
 - Atau, kita dapatkan dari HASH products.

Ketika button diklik, kita tetap gunakan fungsi event, seperti cuplikan kode berikut:

```
event (b1, fn () {  
    var c = get (c2, "text")  
    if (empty(c)) {  
        return c
```

```

}

var x = get(x1, "active")
var xx = ""

if (x) {
    var xx = " dibungkus"
}

var s = get(s1, "contents")
var m = "Tambah " + c + " sebanyak " + s + xx
message(m)
})

```

Walaupun sedikit lebih panjang, prinsipnya mirip dengan contoh-contoh sebelumnya:

- **Diwarnai merah:** kita dapatkan menu yang dipilih (get “text” dari combobox). Apabila kosong (cek dengan fungsi empty), maka kita tidak akan melanjutkan (return fungsi, kembali ke kode pemanggil).
- **Diwarnai biru:** untuk checkbox, kita get “active”. Apabila true (kemungkinan lain adalah false), maka kita akan siapkan sebuah STRING “ dibungkus” (variabel xx, nilai awal adalah STRING “”).
- **Diwarnai hijau:** kita mendapatkan “contents” dari spin. Selanjutnya, kita tinggalkan gabungan STRING, assign ke variabel m, dan tampilkan dengan fungsi message.
- Sekali lagi: Anda mungkin ingin menggunakan nama variabel yang lebih deskriptif (selama valid, lihatlah kembali catatan pada contoh 3).

Walau agak panjang dibandingkan dengan contoh sebelumnya, pada akhirnya pembahasan pun selesai. Apabila Anda perhatikan isi kode program selengkapnya, semua pembahasan telah tercakup, baik di contoh ini, ataupun pada contoh-contoh sebelumnya.

```
var products = {  
    "makanan": ["Bakwan", "Singkong", "Pangsit",  
    "Perkedel"],  
    "minuman": ["Air", "Kopi", "Teh"]  
}
```

```
reset()
```

```
var a1 = component("label", "Kategori")
```

```
var a2 = component("label", "Menu")
```

```
var a3 = component("label", "Jumlah")
```

```
var c1 = component("combobox", ",Makanan,Minuman")
```

```
var c2 = component("combobox", "")
```

```
var s1 = component("spin", "1,1,20,1")
```

```
var x1 = component("checkbox", "Bungkus")
```

```
var b1 = component("button", "Tambah")
```

```
event(c1, fn() {
```

```
    var c = lower(get(c1, "text"))
```

```
    var p = products[c]
```

```
    if (p == null) {
```

```
        var p = []
```

```

    }
    config(c2, "contents", p)
  })
  event(b1, fn() {
    var c = get(c2, "text")
    if (empty(c)) {
      return c
    }
    var x = get(x1, "active")
    var xx = ""
    if (x) {
      var xx = " dibungkus"
    }
    var s = get(s1, "contents")
    var m = "Tambah " + c + " sebanyak " + s + xx
    message(m)
  })

```

```

var g = component("grid", "")
grid_add(g, a1, 0, 0, 1, 1, 0, 1, 1, 1)
grid_add(g, c1, 1, 0, 2, 1, 1, 1, 3, 1)
grid_add(g, a2, 0, 1, 1, 1, 0, 1, 1, 1)
grid_add(g, c2, 1, 1, 2, 1, 1, 1, 3, 1)

```

```
grid_add(g, a3, 0, 2, 1, 1, 0, 1, 1, 1)
grid_add(g, s1, 1, 2, 1, 1, 1, 1, 3, 1)
grid_add(g, x1, 2, 2, 1, 1, 1, 1, 1, 0)
grid_add(g, b1, 2, 3, 1, 1, 1, 1, 0, 2)
add(g)
```

```
size(400, 200)
title("Tambah Pesanan")
show()
```


Contoh 7: Menyimpan ke file teks (+9 baris dari contoh 6)

Pada contoh 6, kita hanya menampilkan pesan bahwa pesanan telah ditambahkan. Bagi pengguna program, informasi ini tersampaikan. Namun, bagi sistem, tidak ada yang tercatat.

Untuk kebutuhan nyata, mungkin hasil entri seperti ini perlu dapat disimpan secara lebih permanen. Kita dapat memanfaatkan sistem database relasional untuk kebutuhan tersebut, namun, untuk contoh ini, kita akan menggunakan file teks biasa.

Kita perlu memikirkan bagaimana tambahan pesanan tersebut disimpan ke file teks:

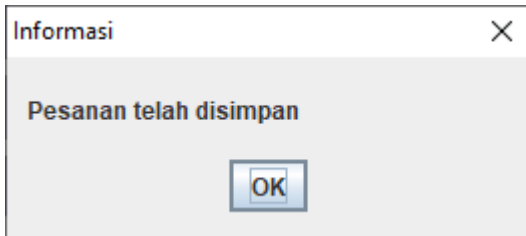
- Kita tidak mencatat siapa yang memesan. Oleh karena itu, semua pesanan akan disimpan ke dalam satu file yang sama, katakanlah dengan nama file pesanan.txt di direktori/folder aktif (misalnya: sama dengan lokasi Singkong.jar).
- Kita akan menyimpan tanggal dan waktu ketika pesanan dibuat. Di Singkong, tanggal dan waktu dapat menggunakan tipe DATE. Ketika akan disimpan ke file teks, nantinya akan berupa STRING.
- Untuk setiap pesanan, mari kita simpan sebagai satu baris tersendiri, yang diakhiri dengan carriage return dan line feed. Di Singkong, untuk mendapatkan carriage return, kita menggunakan fungsi cr. Untuk line feed, kita menggunakan fungsi lf. Tersedia pula fungsi crlf untuk mempersingkat.
- Form kita terdiri atas Kategori, Menu, Jumlah, dan apakah dibungkus atau tidak. Dengan demikian, terdapat 4 field. Untuk Kategori, Menu, dan Jumlah jelas adanya. Untuk apakah dibungkus, kita dapat menyimpannya sebagai true atau false. Atau 1 dan 0 supaya lebih ringkas. Untuk

mendapatkan 1 dari true dan 0 dari false, kita dapat menggunakan fungsi `number_boolean`.

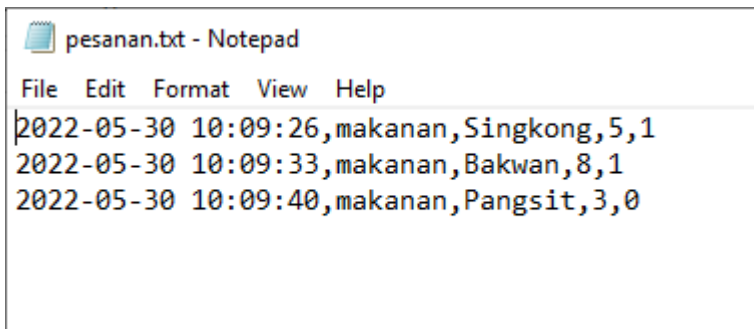
- Kita perlu memberi tanda pemisah antar field. Untuk contoh ini, mari kita sepakati dengan karakter koma (,).
- Digabung dengan tanggal dan waktu pesanan dibuat, mari kita sepakati bahwa setiap pesanan akan disimpan sebagai:

(tanggal/waktu),kategori,menu,jumlah,(1 atau 0),(cr)(lf)

Anda mungkin berpendapat: cukup sampai di sini pengantarnya! Baiklah kalau begitu. Mari kita lihat dulu apa yang akan tampil ketika pesanan berhasil disimpan ke file `pesanan.txt`.



Dan, berikut adalah contoh isi file `pesanan.txt` (dibuka di Notepad Windows 10) ketika pesanan ditambahkan beberapa kali (oleh penyuka Singkong, Bakwan, dan Pangsit).



Dari cuplikan isi file tersebut, kita bisa mengartikan:

- Terdapat 3 kali pesanan ditambahkan.
- Pesanan pertama adalah Singkong sebanyak 5 porsi dibungkus.
- Pesanan kedua adalah Bakwan sebanyak 8 porsi dibungkus.
- Pesanan ketiga adalah Pangsit 3 porsi dimakan di tempat.

Apabila yang memesan semua ini adalah orang yang sama, maka sambil menunggu Singkong dan Bakwan disiapkan, beliau bisa makan 3 porsi pangsit di tempat.

Mari kita lihat penambahan 9 baris tersebut (8 dan 1 baris kosong), yang diwarnai hijau:

```

var m = "Tambah " + c + " sebanyak " + s + xx
message(m)

var k = lower(get(c1, "text"))
var d = [0, k, c, s, number_boolean(x)]
var d = join(", ", d) + crlf()
if (append("pesanan.txt", d)) {
    message("Pesanan      telah      disimpan",
"Informasi")
} else {
    message("Pesanan      gagal      disimpan",
"Kesalahan")
}

```

Sebagaimana dapat dilihat, kita menambahkan baris baru ini pada event handler ketika button Tambah ditekan, setelah baris message(m).

Penjelasan kode program:

- Pada baris var d yang pertama, kita membuat sebuah ARRAY yang terdiri dari semua field yang ingin disimpan. Tanggal dan waktu saat ini diwakili dengan @. Pada baris var d yang kedua, kita menggabungkan masing-masing elemen dalam ARRAY dengan sebuah koma (,), sehingga hasil akhirnya berupa sebuah STRING, yang lalu kita tambahkan dengan hasil pemanggilan fungsi crlf.
 - o Di sini, kita melihat bahwa ARRAY mengandung elemen-elemen dengan tipe yang berbeda. Ada DATE, STRING dan NUMBER.
 - o Dengan fungsi join, pada akhirnya, semua akan diubah ke STRING.
 - o Setiap tipe di Singkong memiliki representasi STRING dari nilainya. Sebagai contoh, @ yang merupakan tanggal dan jam saat ini, akan memiliki representasi STRING berupa tanggal dan waktu dalam format tahun (4 digit)-bulan (2 digit)-tanggal (2 digit)(spasi)(jam 2 digit):(menit 2 digit):(detik 2 digit).
- Fungsi append akan menambahkan konten ke sebuah file teks. Apabila berhasil, fungsi ini mengembalikan true. Oleh karenanya, kita akan cek dan tampilkan pesan yang sesuai. Sebelumnya kita telah membahas penggunaan if. Di sini, dicontohkan juga penggunaan else pada if.

Contoh 8: Simpan dan tampilkan file CSV (+10 baris dari contoh 6)

Pada contoh 7 sebelumnya, secara sekilas, kita mencapai apa yang diinginkan. Isi file pesanan.txt dapat ditambahkan sesuai pesanan yang dibuat.

Tapi, mari kita lihat lebih lanjut.

Pertama, kita menambahkan setiap field ke sebuah ARRAY, barulah setiap elemen dalam ARRAY tersebut digabungkan dengan sebuah karakter pemisah koma (kemudian ditambahkan hasil pemanggilan fungsi `crlf` dan di-append ke file).

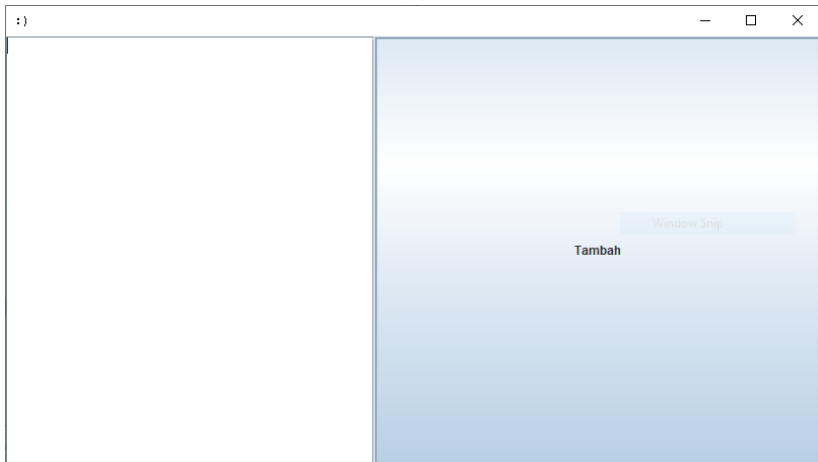
Sejauh ini, tidak menjadi masalah karena nilai setiap field (misal: 2022-05-30 10:09:26 atau makanan atau Singkong atau 5 atau 1) tidaklah mengandung karakter pemisah tersebut. Dalam perkembangan, bagaimana kalau nama menu makanan mengandung karakter pemisah? Atau, bagaimana kalau pelanggan bisa menambahkan catatan tertentu yang mengandung karakter pemisah (misal: tolong, tambahkan lebih banyak kalori).

Kedua, sejauh ini kita mengasumsikan setiap pesanan dituliskan sebagai sebuah baris tersendiri (secara teknis: diakhiri oleh keluaran dari pemanggilan fungsi `crLf`, yang merupakan akhir baris file teks di Windows misalnya). Sekarang, apabila pelanggan bisa menuliskan karakter koma (atau karakter pemisah lain), ada kemungkinan pelanggan bisa menambahkan karakter yang terkandung dalam keluaran dari fungsi `crLf` tersebut. Misal, ketika program mengizinkan catatan yang lebih dari satu baris (COMPONENT edit) dan pengguna menekan ENTER. Sebagai contoh, catatannya adalah:

Tolong, tambahkan lebih banyak kalori
--

Maka, apabila ditambahkan ke file apa adanya, per pesanan tidak lagi berupa satu baris. Sekedar menambahkan ke file tentu tidak masalah. Membaca file tersebut akan perlu usaha lebih.

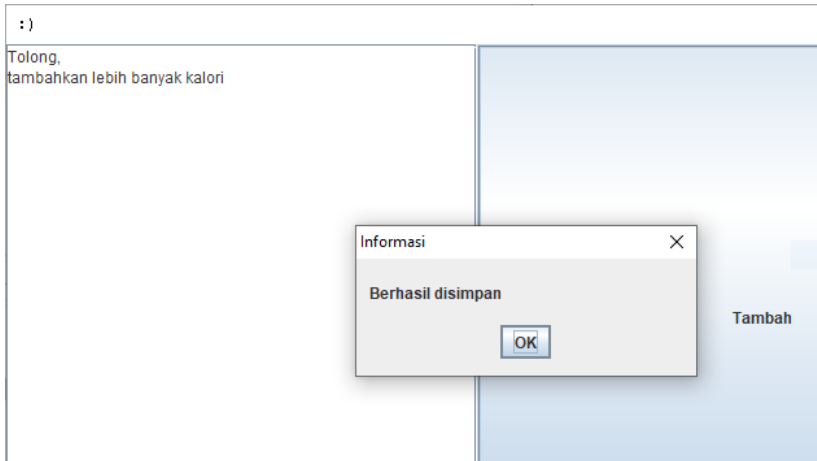
Mari kita buat program sederhana supaya apa yang kita bahas ini terlihat lebih konkrit:



Apabila pada COMPONENT edit di sisi kiri kita isikan empat kali sesuai teks berikut:

- Halo
- tolong, tambahkan lebih banyak kalori
- Tolong,<tekan ENTER>
tambahkan lebih banyak kalori
- Tes

Lalu masing-masing diikuti penekanan tombol Tambah, dengan contoh dengan penekanan ENTER terlihat seperti pada gambar berikut:



Maka berikut adalah isi file yang dihasilkan (disimpan pada file test.txt):

```
test.txt - Notepad
File Edit Format View Help
2022-05-31 19:55:56,Halo
2022-05-31 19:56:10,tolong, tambahkan lebih banyak kalori
2022-05-31 19:56:39,Tolong,
tambahkan lebih banyak kalori
2022-05-31 20:01:27,Tes
```

Apa yang bisa kita cermati dari isi file tersebut?:

- Kita mungkin mengharapkan nilai ideal seperti baris pertama dan baris terakhir, dalam artian per baris terdiri dari dua field, yaitu tanggal/jam dan teks, yang dipisahkan koma.
- Kenyataannya, di baris kedua, dengan mempertimbangkan jumlah koma, jumlah field adalah tiga: 2022-05-31 19:56:10, tolong, dan tambahkan lebih banyak kalori.

- Dan, di baris ketiga, juga dengan mempertimbangkan jumlah koma, jumlah field juga tetap tiga: 2022-05-31 19:56:39, Tolong, dan teks kosong.
- Lebih repot lagi: kita mengisi 4 kali, namun mendapatkan 'seolah' adanya 5 baris (sebagaimana di gambar). Baiklah, apabila editor teks yang Anda gunakan tidak tampil sebagaimana dalam gambar terakhir, ketika kode program yang akan tampil nanti dijalankan, mari kita lihat visualisasi line ending dengan editor vim (vim -b dan :set list, tipe file terdeteksi unix) berikut:

```
2022-05-31 19:55:56,Halo^M$
2022-05-31 19:56:10,tolong, tambahkan lebih banyak kalori^M$
2022-05-31 19:56:39,Tolong,$
tambahkan lebih banyak kalori^M$
2022-05-31 20:01:27,Tes^M$
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
test.txt [unix] (20:01 31/05/2022)
:set list
```

- Bisa dilihat di gambar di atas, carriage return (tampil sebagai ^M) dan line feed (tampil sebagai \$) adalah akibat kita memanggil fungsi crlf. Tetap 4 kali bukan? Tapi penekanan ENTER di COMPONENT edit menambahkan sebuah line feed di baris 3.

- Menulis ke file saja pun sudah report. Bagaimana kita membaca file tersebut?

Berikut adalah kode program uji coba tersebut:

```
reset()
var e = component("edit", "")
var b = component("button", "Tambah")
add([e, b])
show()

event(b, fn() {
    var c = get(e, "contents")
    var d = [e, c]
    var d = join(", ", d) + crlf()
    if (append("test.txt", d)) {
        message("Berhasil disimpan", "Informasi")
    } else {
        message("Gagal disimpan", "Kesalahan")
    }
})

show()
```

Tidak banyak beda dengan yang dibahas pada contoh sebelumnya, bukan?

Baiklah. Kita mungkin berlebihan di sini, karena bahkan dalam form pesanan kita, tidak disediakan cara untuk menambahkan catatan.

Tapi, semua kerepotan dan pembahasan bertele-tele yang kita awali di contoh 8 ini tentu saja bertujuan baik. Setidaknya, kita jadi melihat bahwa kode program yang kita tulis di contoh 7 sebelumnya tidak berlaku untuk kasus yang lebih umum.

Sebelum kita lanjut, mari kita coba baca dan pisahkan field dalam file test.txt tersebut. Aktiflah terlebih dahulu di tab Interactive di Singkong. Dengan asumsi test.txt berada di direktori aktif, ketikkanlah kode-kode berikut (tanpa mengetikkan >) dan tekanlah ENTER atau klik tombol Eval setiap kalinya:

```
> var c = read("test.txt")
> c
"2022-05-31 19:55:56,Halo
2022-05-31 19:56:10,tolong, tambahkan lebih banyak
kalori
2022-05-31 19:56:39,Tolong,
tambahkan lebih banyak kalori
2022-05-31 20:01:27,Tes
"
```

Sampai di sini, cuplikan keluaran di atas memperlihatkan bahwa kita berhasil membaca file test.txt dengan sekali panggil fungsi read. Karena sebelumnya kita menambahkan setiap baris dengan keluaran dari fungsi crlf, mari kita pisahkan kembali (mengggunakan fungsi split) berdasarkan pemisah yang sama:

```
> var c = split(c, crlf())
> c
["2022-05-31      19:55:56,Halo",      "2022-05-31
19:56:10,tolong,  tambahkan lebih banyak kalori",
"2022-05-31 19:56:39,Tolong,
tambahkan lebih banyak kalori",  "2022-05-31
20:01:27,Tes"]
```

```
> len(c)
```

```
4
```

Bisa dilihat, kita mendapatkan tetap 4 baris (sebagaimana penggunaan fungsi len pada ARRAY). Beberapa barisnya adalah sebagai berikut:

```
> c[0]
```

```
"2022-05-31 19:55:56,Halo"
```

```
> c[1]
```

```
"2022-05-31 19:56:10,tolong, tambahkan lebih banyak
kalori"
```

```
> c[2]
```

```
"2022-05-31 19:56:39,Tolong,
tambahkan lebih banyak kalori"
```

Sekarang, untuk baris pertama (index 0), memisahkan berdasarkan karakter pemisah koma tidaklah sulit:

```
> split(c[0], ",")
["2022-05-31 19:55:56", "Halo"]
```

Kita mendapatkan dua field:

```
> len(split(c[0], ","))
2
```

Tapi bagaimana dengan baris kedua (indeks 1)?

```
> split(c[1], ",")
["2022-05-31 19:56:10", "tolong", " tambahkan lebih
banyak kalori"]
```

Tiga field bukan? Sama dengan baris ketiga (index 2):

```
> split(c[2], ",")
["2022-05-31 19:56:39", "Tolong", "
tambahkan lebih banyak kalori"]
```

Hanya, sebagai bonus, kita mendapatkan line feed.

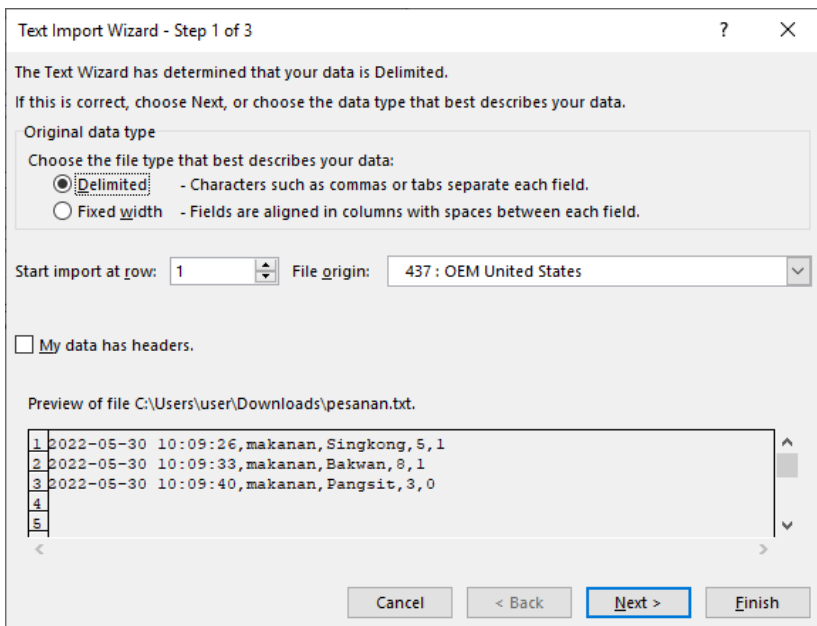
Sampai di sini, berdasarkan contoh-contoh kode tersebut, bisa kita lihat, kita memang masih mendapatkan 4 baris dan bukannya 5, karena line ending yang kita gunakan adalah carriage return diikuti line feed, sementara penekanan ENTER pada COMPONENT edit menambahkan line feed.

Tapi, bagaimana dengan jumlah field antara 2 dan 3 tersebut? Ini akan lebih menjadi kendala.

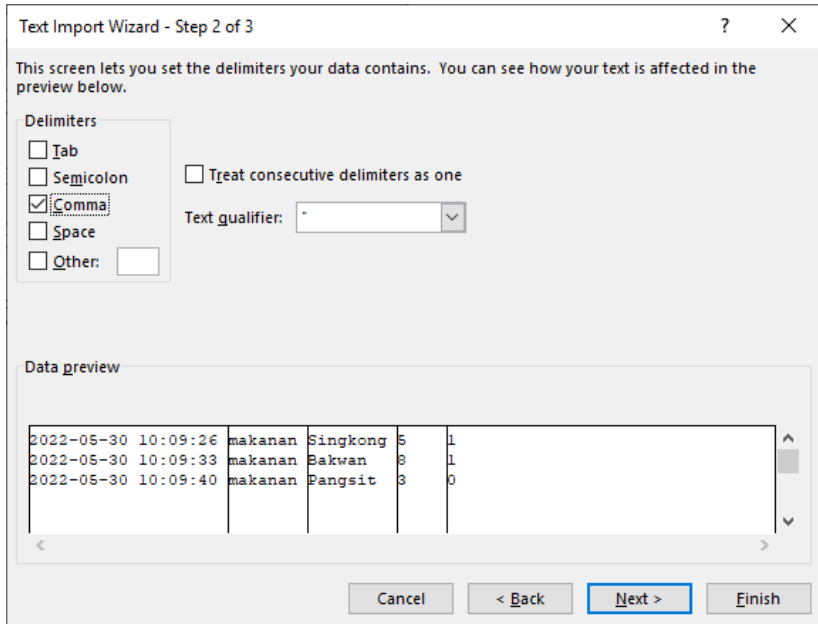
Belum lagi, setelah menghabiskan 8 halaman di contoh 8 ini, kita bahkan belum membahas banyak. (Apa? Belum cukup banyak?)

Apa yang coba kita sampaikan adalah, pada contoh 8, kita mencoba menghasilkan file teks dalam format khusus yang umum dikenal sebagai Comma-Separated Values (CSV, yang didefinisikan dalam RFC 4180). Setiap baris data (atau record) dapat terdiri dari satu atau lebih field, dipisahkan karakter tertentu, atau koma (seperti nama format filenya).

File ini bisa kita buka dengan nyaman di berbagai spreadsheet, sebagai contoh ketika membuka file pesanan.txt (dari contoh 7) dengan Excel:



Kliklah tombol Next.



Pilihlah Comma sebagai delimiters dan kliklah tombol Finish.

	A	B	C	D	E
1	30/05/2022 10:09	makanan	Singkong	5	1
2	30/05/2022 10:09	makanan	Bakwan	8	1
3	30/05/2022 10:09	makanan	Pangsit	3	0
4					
5					

Bisa kita lihat, sesuai seperti yang kita harapkan.

Bagaimana kalau kita mencoba membuat viewernya dengan bahasa Singkong? Untuk file pesanan.txt, tidaklah sulit. Perhatikanlah kode program di halaman berikut, yang hanya terdiri dari 9 baris:

```

reset ()

var t = component ("table",
"Waktu, Kategori, Menu, Jumlah, Bungkus")

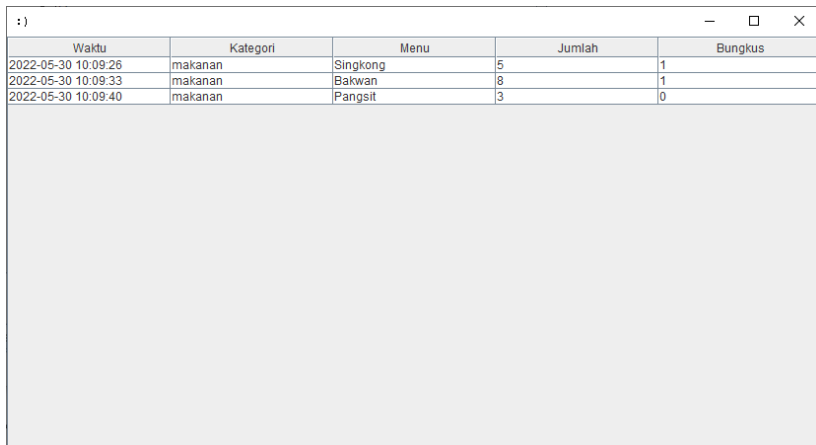
add (t)

show ()

var c = split (read ("pesanan.txt"), crlf ())
each (c, fn (e, i) {
    table_add (t, [split (e, ",")])
})

```

Dan, ketika dijalankan:



Waktu	Kategori	Menu	Jumlah	Bungkus
2022-05-30 10:09:26	makanan	Singkong	5	1
2022-05-30 10:09:33	makanan	Bakwan	8	1
2022-05-30 10:09:40	makanan	Pangsit	3	0

Dengan mengubah kode program tersebut menjadi:

```

reset ()

var t = component ("table", "Waktu, Komentar")

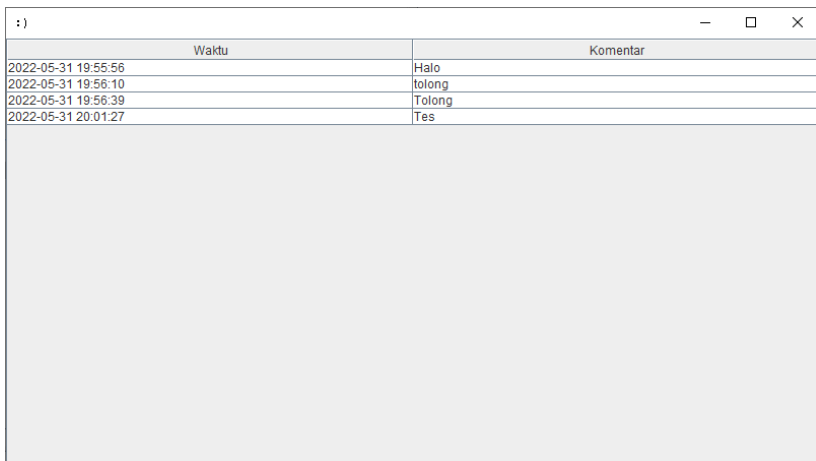
add (t)

```

```
show()
```

```
var c = split(read("test.txt"), crlf())  
each(c, fn(e, i) {  
    table_add(t, [split(e, ",")])  
})
```

Ketika dijalankan untuk membaca test.txt, tampilannya adalah sebagai berikut:



Waktu	Komentar
2022-05-31 19:55:56	Halo
2022-05-31 19:56:10	tolong
2022-05-31 19:56:39	Tolong
2022-05-31 20:01:27	Tes

Lihatlah, permintaan untuk menambahkan lebih banyak kalori tersebut sudah ditambahkan.

Sampai di sini, untuk bekerja dengan file CSV di Singkong, lebih disarankan untuk menggunakan modul bawaan dengan nama csv, yang diantaranya:

- Akan menambahkan kutip ganda apabila sebuah field mengandung karakter pemisah. Jadi, apabila pemisah yang digunakan adalah koma dan isinya juga mengandung koma,

maka kutip ganda secara otomatis akan digunakan. Misal, untuk: tolong, tambahkan lebih banyak kalori, maka akan menjadi “tolong, tambahkan lebih banyak kalori”.

- Karakter newline dalam field juga didukung, sehingga contoh penekanan ENTER pada Tolong, <ENTER> tambahkan lebih banyak kalori tidak akan menjadi kendala.

Mari kita sesuaikan kembali program uji coba sebelumnya yang akan menambahkan ke file test-csv.txt, namun dengan modul bawaan csv:

```
load_module("csv")
reset()
var e = component("edit", "")
var b = component("button", "Tambah")
add([e, b])
show()

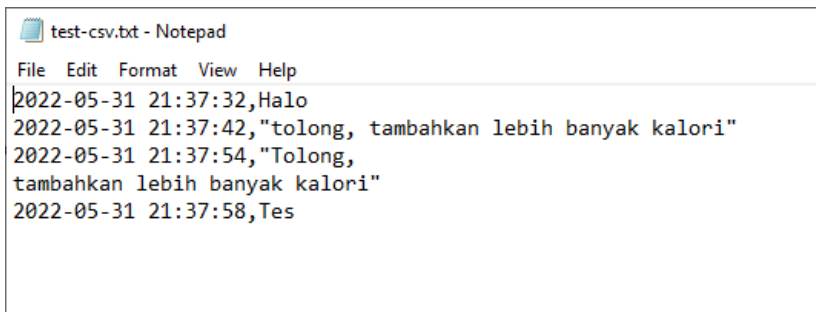
event(b, fn() {
    var c = get(e, "contents")
    var d = [e, c]
    var d = csv_to_string(",", [d]) + crlf()
    if (append("test-csv.txt", d)) {
        message("Berhasil disimpan", "Informasi")
    } else {
        message("Gagal disimpan", "Kesalahan")
    }
}
```

```
})
```

```
show()
```

Bisa kita lihat, perbedaannya hanyalah pada baris pertama (`load_module`) dan penggunaan fungsi `csv_to_string` dari modul tersebut. Perhatikanlah kita tidak perlu manual membuat ARRAY dan menggabungkan setiap elemen dalam ARRAY sebagai sebuah STRING.

Sekarang, mari kita lihat `test-csv.txt`:



```
test-csv.txt - Notepad
File Edit Format View Help
2022-05-31 21:37:32,Halo
2022-05-31 21:37:42,"tolong, tambahkan lebih banyak kalori"
2022-05-31 21:37:54,"Tolong,
tambahkan lebih banyak kalori"
2022-05-31 21:37:58,Tes
```

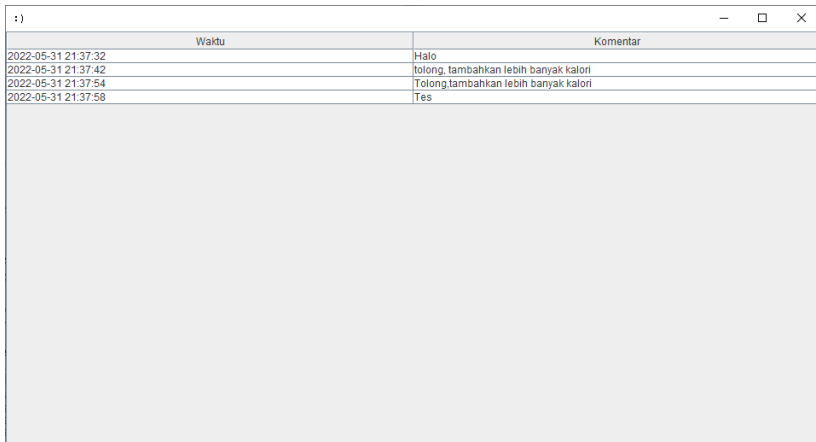
Terlepas dari waktu yang berbeda, bisa kita lihat, selama kutip ganda diperlukan, maka secara otomatis akan ditambahkan.

Dan, dengan modul `csv` yang sama, namun dengan fungsi berbeda, mari kita sesuaikan program viewernya:

```
load_module("csv")
reset()
var t = component("table", "Waktu,Komentar")
add(t)
show()
```

```
var a = csv_from_string(",", read("test-csv.txt"))
config(t, "contents", a)
```

Selain kode programnya lebih pendek, kita pun dapat menampilkannya dengan benar, seperti gambar berikut:



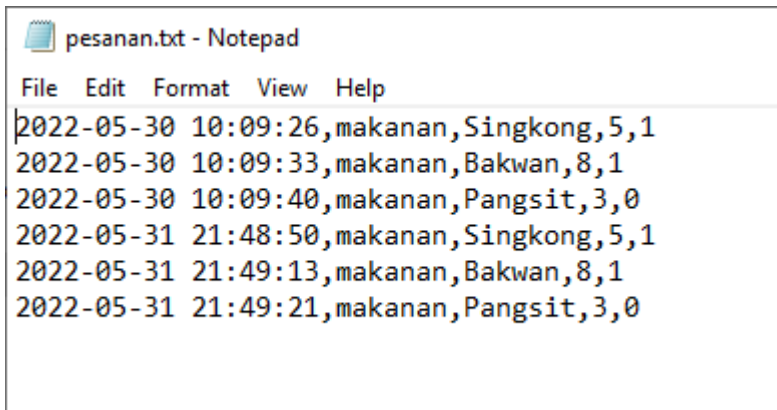
Waktu	Komentar
2022-05-31 21:37:32	Halo
2022-05-31 21:37:42	tolong, tambahkan lebih banyak kalori
2022-05-31 21:37:54	Tolong tambahkan lebih banyak kalori
2022-05-31 21:37:58	Tes

Dengan prinsip serupa, tidaklah sulit untuk mengubah kode program pada contoh 6 sebelumnya untuk menggunakan modul csv. Mari kita lihat perbedaannya (diff): file pertama test.singkong adalah dari contoh 7 dan test-csv.singkong adalah yang telah disesuaikan dengan menggunakan modul csv:

```
$ diff test.singkong test-csv.singkong
0a1
> load_module("csv")
40c41
<     var d = join(",", d) + crlf()
---
>     var d = csv_to_string(",", [d]) + crlf()
```

Bisa kita lihat, yang bertambah adalah baris `load_module` dan yang berubah adalah yang tadinya secara manual dengan fungsi `join`, kini lebih nyaman dengan fungsi `csv_to_string` dari modul bawaan `csv`.

Memang, karena tidak adanya karakter pemisah dan `newline` dalam field, isi file `pesanan.txt` kita tidak berbeda (3 baris terakhir menggunakan modul bawaan `csv`):



```
pesanan.txt - Notepad
File Edit Format View Help
2022-05-30 10:09:26,makanan,Singkong,5,1
2022-05-30 10:09:33,makanan,Bakwan,8,1
2022-05-30 10:09:40,makanan,Pangsit,3,0
2022-05-31 21:48:50,makanan,Singkong,5,1
2022-05-31 21:49:13,makanan,Bakwan,8,1
2022-05-31 21:49:21,makanan,Pangsit,3,0
```

Catatan: modul bawaan `csv` ditulis sepenuhnya dengan bahasa pemrograman Singkong. Apabila ingin melihat kode atau cara kerjanya, Anda dapat meng-extract file `csv.singkong` dari direktori `singkong` dalam file `Singkong.jar`.

Contoh 9: Pesanan makanan berbeda untuk setiap meja (66 baris)

Pada form sebelumnya, setiap penambahan pesanan akan ditambahkan ke file pesanan.txt yang sama, tanpa data siapa yang menambahkan. Walaupun ini mungkin cukup bagi yang memasak, form tersebut jelas tidak mengakomodir kebutuhan yang mengantar ke meja ataupun kasir.

Tapi, sebelum kita menulis beberapa baris kode baru untuk kebutuhan tambahan, mari kita pikirkan ulang form tersebut.

Anggap saja ini adalah sebuah rumah makan dengan jumlah meja terbatas, dimana pelanggan bisa memesan untuk makan di tempat ataupun dibawa. Agar program yang kita buat tetap sederhana, untuk makan di tempat, data siapa yang menambahkan tersebut bisa berupa nomor meja. Untuk yang dibawa, kita bisa memberikan nomor meja tersendiri disertai informasi tambahan (nama atau lainnya) apabila perlu (namun tidak dibahas dalam contoh ini).

Dengan demikian, sebelum form tambah pesanan tersebut ada, kita perlu menyiapkan form siapa saja yang dapat memesan. Apabila rumah makan kita terdiri dari 15 meja, mari kita siapkan 15 button, yang akan berwarna tertentu apabila belum ditempati, dan warna lain apabila sebaliknya. Jadi, setiap meja memiliki status tersendiri. Mari kita siapkan struktur datanya:

```
var tables = {  
    "01": [0, 0],  
    "02": [1, 0],  
    "03": [2, 0],  
    "04": [3, 0],
```

```
"05": [4, 0],
"06": [0, 1],
"07": [1, 1],
"08": [2, 1],
"09": [3, 1],
"10": [4, 1],
"11": [0, 2],
"12": [1, 2],
"13": [2, 2],
"14": [3, 2],
"15": [4, 2]
}
```

Sebuah HASH. Kenapa tidak ARRAY, Anda mungkin bertanya? Bukankah dengan HASH, kode menjadi lebih panjang dan kita perlu menentukan nama meja secara manual seperti dalam kode tersebut?

Tentu saja, bukan tujuan kami agar kode menjadi lebih panjang. Dengan HASH, kita bisa memberikan nama meja yang bukan hanya berupa indeks (dari 0, sebagaimana halnya ARRAY). Dengan demikian, nama meja kita bisa berupa STRING seperti "A". Dan, kita pun bisa menentukan posisi tombol dalam grid. Kita ingin layout button semirip mungkin dengan kondisi nyatanya.

Lalu, karena setiap meja memiliki status apakah sedang ditempati atau tidak, mari kita siapkan juga sebuah HASH berikut:

```
var table_status = {}
```

Begitu saja? Satu baris?

Benar. Key dari `table_status` tersebut akan berupa nama meja. Valuenya akan berupa `NUMBER` seperti 0 (kosong) dan 1 (ditempati) untuk saat ini, namun ke depan bisa saja `NUMBER` lain misal untuk status telah dipesan ataupun lainnya. Karena saat ini berupa `HASH` kosong, seperti Anda mungkin telah menduganya, kita akan membuat pemetaan baru (nama meja ke status) secara dinamis.

Sembilan baris kode berikut diantaranya digunakan untuk menambahkan `button-button` ke `grid`, sesuai dengan posisi terdefinisi dalam variabel `tables`. Anda mungkin tidak perlu mengetikkan kodenya karena akan kita ubah berikutnya. Kami tahu, ini mungkin kurang menyenangkan (bagi Anda yang tidak menuruti himbuan tersebut), namun tujuannya supaya kita bisa membahas dari yang sesederhana mungkin.

```
reset()

var g = component("grid", "")

each(keys(tables), fn(e, i) {
    var v = tables[e]
    var b = component("button", e)
    grid_add(g, b, v[0], v[1], 1, 1, 1, 1, 3, 0)
})

add(g)

show()
```

Bisa kita lihat, tidak ada kejutan. Setidaknya, kita sudah pernah bekerja dengan kode serupa.

Baiklah. Mungkin ada sedikit kejutan. Kalau tidak, pasti kami tidak akan memberikan warna tersendiri. Mari kita lihat yang paling sederhana dulu, yaitu fungsi `keys`, yang akan mengembalikan semua

key dalam sebuah HASH. Aktiflah di tab Interactive dan ketikkanlah kode berikut (seperti biasa, hanya yang diawali >, tanpa mengetikkan >):

```
> var h = {"nama": "Singkong", "umur": 3}
```

```
> h
```

```
{"nama": "Singkong", "umur": 3}
```

```
> keys(h)
```

```
["nama", "umur"]
```

```
> values(h)
```

```
["Singkong", 3]
```

```
> h["nama"]
```

```
"Singkong"
```

```
> h["umur"]
```

```
3
```

Kita menggunakan `keys` untuk mendapatkan semua key (dalam hal ini: "nama" dan "umur") dan `values` untuk mendapatkan semua value (dalam hal ini, "Singkong" dan 3). Untuk mendapatkan value dari sebuah key, kita gunakan operator indeks `[dan]` seperti contoh tersebut (dengan key "nama", valuenya adalah "Singkong").

Kembali ke variabel `tables`, pemanggilan `keys` (diwarnai biru) akan mengembalikan nama-nama meja dalam `STRING`.

Fungsi `each` telah digunakan dalam contoh sebelumnya, namun tidak dibahas. Pada dasarnya, dengan fungsi `each`, untuk setiap elemen dalam `ARRAY` (`keys` akan mengembalikan `ARRAY`), kita panggil sebuah `FUNCTION` (yang diwarnai merah). `FUNCTION` tersebut harus memiliki dua argumen, yaitu elemen itu sendiri dan indeks nya (dari 0). Kembalilah ke tab `Interactive` dan ketikkanlah kode berikut (dalam satu baris):

```
> each(keys(h), fn(e, i) {println("Elemen: " + e +  
", indeks: " + i)})
```

```
Elemen: nama, indeks: 0
```

```
Elemen: umur, indeks: 1
```

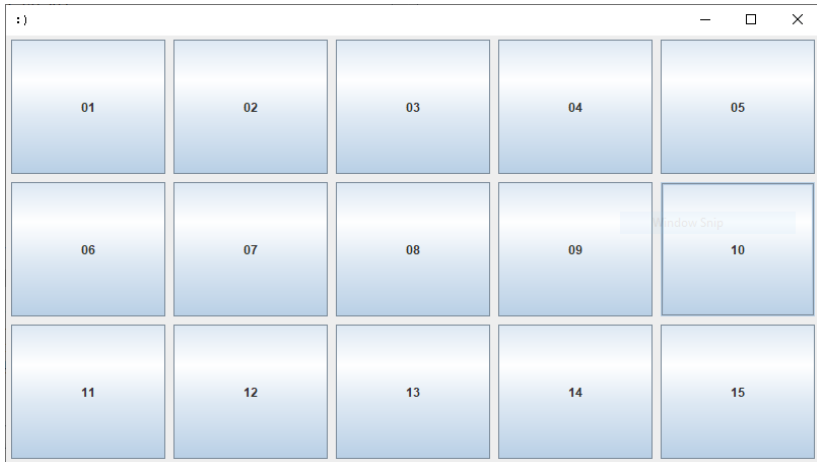
Message box akan ditampilkan, namun pada akhirnya keluaran dari kode tersebut adalah sebagaimana tertulis di atas. Apabila terdapat `ARRAY` ["nama", "umur"], maka elemen-elemennya adalah "nama" (dengan indeks 0) dan "umur" (indeks 1).

Dengan demikian, argumen `e` dalam kode yang diwarnai merah adalah nama meja. Sementara, variabel `v` akan berupa `value`, yaitu posisi `button` (`x`, `y`) dalam `grid`, sebagai `ARRAY`. Selbihnya, kita sudah pernah membahas pembuatan `COMPONENT` `button` dan penambahan ke `grid`.

Terkait sintaks, pemanggilan fungsi `each` sendiri (diwarnai hijau) tentu membutuhkan kurung buka (dan kurung tutup). Adanya `}` pada baris terakhir tentu dapat dipahami sebagai:

- `}` adalah penutup blok `FUNCTION` yang dibuat
- `)` adalah penutup pemanggilan fungsi `each`.

Ketika kode tersebut dijalankan, frame akan tampil seperti gambar di halaman berikut.



Untuk setiap meja yang dimiliki memang telah diwakili dengan sebuah button. Tapi, berhubung semua meja masih belum ditempati, sesuai kesepakatan kita, harusnya semua button tersebut berwarna tertentu, bukan? Kenyataannya, tidaklah demikian.

Mari kita tambahkan beberapa baris kode.

Di bawah baris `table_status`, kita pastikan bahwa setiap meja masih kosong:

```
each(keys(tables), fn(e, i) {  
    set(table_status, e, 0)  
})
```

Kemudian, kita siapkan kemungkinan status dan warna button dengan sebuah HASH:

```
var button_color = {  
    0: "meja_kosong",
```

```
    1: "meja_terisi"  
}
```

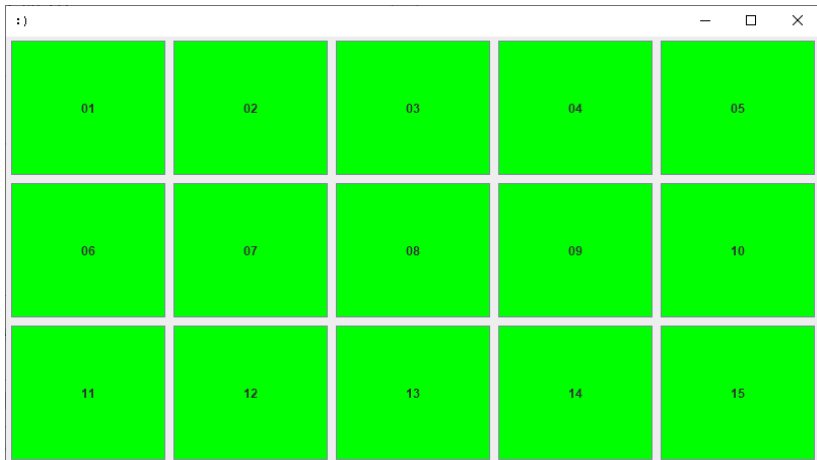
Sebagai sedikit latihan, gantilah tulisan “meja_kosong” dan “meja_terisi” dengan kode warna sesungguhnya, seperti green, ataupun yang didapatkan dari fungsi `color_chooser`. Cobalah.

Key 0 berarti belum ditempati, dan 1 adalah sebaliknya. Dengan demikian, ke depannya, kita bisa menambahkan key dan value (warna dalam STRING) lain.

Lalu, dalam FUNCTION ketika memanggil `each`, kita tambahkan 3 baris kode berikut setelah pemanggilan `grid_add`:

```
var s = table_status[e]  
var c = button_color[s]  
config(b, "background", c)
```

Berikut adalah frame kita ketika kode dijalankan (green sebagai `meja_kosong`):



Cukup menyilaukan. Tapi, ini menandakan semua meja masih kosong.

Tidak sabar ingin melihat bagaimana kalau beberapa meja telah ditempati? Sebelum baris reset, tambahkan dua baris berikut, yang kemudian dapat dikomentari atau dihapus:

```
set(table_status, "01", 1)
```

```
set(table_status, "02", 1)
```

Jalankan kembali program (pastikanlah warna untuk meja_terisi telah disesuaikan dengan kode warna ataupun kembalian dari fungsi `color_chooser`).

Kita akan melihat, meja yang terisi (meja "01" dan "02") diwarnai sesuai warna yang didefinisikan untuk warna meja terisi, dan selebihnya diwarnai sesuai warna meja kosong.

Masing-masing meja diwakili dengan sebuah button, yang tentu saja dapat ditekan. Sayangnya, sampai titik ini, kita belum menulis kode sama sekali untuk menangani ketika button-button tersebut ditekan.

Mari kita bayangkan ketika salah satu button tersebut ditekan. Idealnya, untuk meja yang masih kosong, kita perlu menandai bahwa meja tersebut akan ditempati dan siap untuk menerima tambahan pesanan. Sementara, untuk meja yang sudah ditempati, kita dapat menerima tambahan pesanan ataupun menandai meja tersebut akan kosong (misal ketika sudah selesai).

Tapi, untuk menjaga contoh program kita tetap sederhana, berikut adalah apa yang akan kita lakukan:

- Menampilkan daftar pesanan yang ada
- Menambahkan pesanan baru

Mari kita tambahkan beberapa baris kode berikut setelah var g:

```
var g2 = component("grid", "")
var a = component("label", "Meja")
var t = component("table",
"Waktu,Kategori,Menu,Jumlah,Bungkus")
grid_add(g2, a, 0, 0, 1, 1, 0, 0, 1, 1)
grid_add(g2, t, 0, 1, 1, 1, 1, 1, 3, 1)
```

Dan, ubah pemanggilan fungsi add dari:

```
add(g)
```

menjadi:

```
add([g, g2])
```

Berikut adalah tampilannya ketika program dijalankan (warna meja kosong adalah green):



Table di sebelah kanan adalah untuk menampilkan daftar pesanan untuk setiap meja. Mari kita tambahkan aksi ketika button ditekan, setelah baris config:

```
event(b, fn () {  
    config(a, "text", "Meja: " + e)  
})
```

Hanya beberapa baris. Nantinya, kita akan lengkapi lagi. Namun, untuk saat ini, perhatikanlah bahwa kita membuat event handler secara dinamis. Di dalam event handler tersebut, pada dasarnya, kita mengatur text label menjadi sesuai nama meja. Cobalah klik pada meja 08 misalnya, dan tampilannya akan menjadi:



Label Meja di sisi kanan atas akan berubah sebagaimana pada gambar.

Di dalam event handler untuk setiap button, kita belum menampilkan daftar pesanan. Mari kita siapkan terlebih dahulu sebuah HASH, setelah `button_color`, dan buat pemetaan secara dinamis:

```
var table_orders = {}
```

```
each(keys(tables), fn(e, i) {
    set(table_orders, e, [])
})
```

Anda mungkin ingin menambahkan baris uji berikut, yang silahkan dihapus atau dikomentari nantinya:

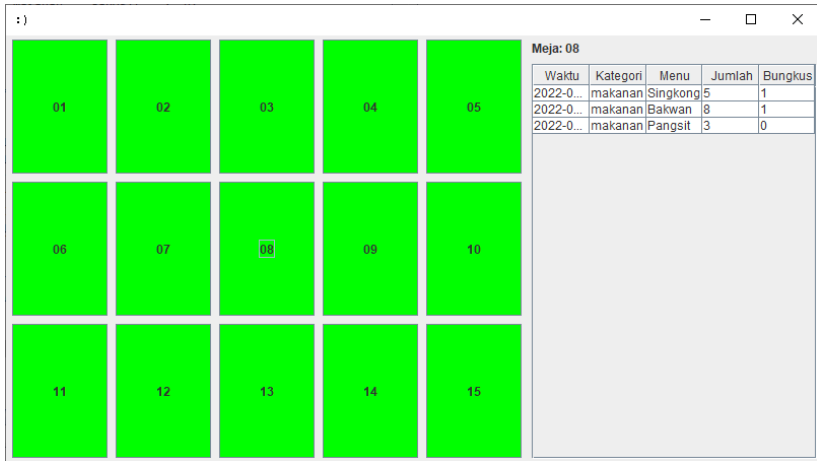
```
set(table_orders, "08", [
    [@, "makanan", "Singkong", 5, 1],
    [@, "makanan", "Bakwan", 8, 1],
    [@, "makanan", "Pangsit", 3, 0]
])
```

Pesanan ini sama dengan contoh pesanan sebelumnya, hanya tanggal dan waktunya berbeda (karena @ adalah tanggal dan waktu ketika kode program dijalankan/dievaluasi).

Kita tambahkan baris berikut dalam event handler button setiap meja, setelah config pada label:

```
config(t, "contents", table_orders[e])
```

Cobalah jalankan kembali program dan kliklah pada meja 08. Perhatikanlah gambar di halaman berikut.



Perbesarlah ukuran frame apabila teks pada table terpotong. Dan perhatikanlah bahwa ketika kita klik pada meja lain, tabel akan kosong dan hanya pada meja 08, daftar pesanan akan tampil. Dengan demikian, kita telah berhasil menampilkan daftar pesanan. Baris kode uji di halaman sebelumnya dapat dikomentari atau dihapus.

Sekarang, bagaimana kita menambahkan pesanan baru sebagaimana contoh-contoh sebelumnya? Mari kita siapkan sebuah ARRAY bantu yang berisikan hanya satu elemen, sebagai penanda meja yang sedang terpilih. Tempatkanlah di bawah pengisian table_orders:

```
var table_active = [""]
```

Setelah itu, tempatkanlah beberapa baris berikut setelah pembuatan COMPONENT table (var t =):

```
var ba = component("button", "Tambah Pesanan")
config(ba, "active", 0)
event(ba, fn() {
    var e = table_active[0]
```



```

    if (!empty(e)) {
        var ee = table_orders[e]
        var ee = ee + [@, "makanan", "Singkong", 1,
1]

        set(table_orders, e, ee)
        config(t, "contents", table_orders[e])
    }
})

```

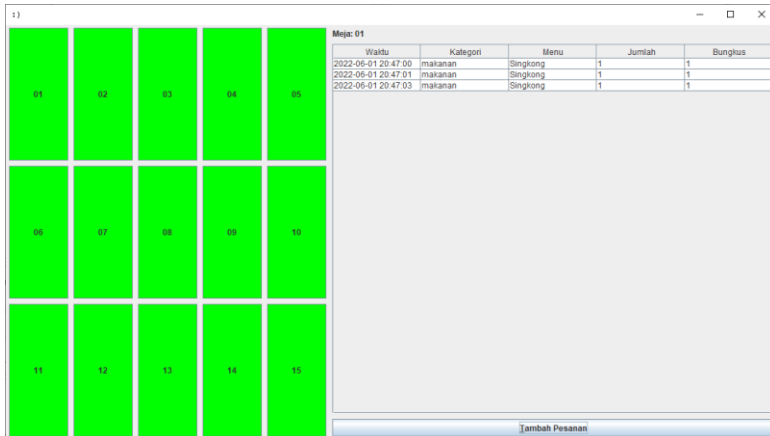
Kemudian, tambahkanlah button ke grid, setelah baris `grid_add g2` terakhir:

```
grid_add(g2, ba, 0, 2, 1, 1, 0, 0, 1, 1)
```

Dan, pada akhirnya, pada event handler button untuk setiap meja, tambahkanlah baris berikut (setelah `config(t, "contents", table_orders[e])`):

```
set(table_active, 0, e)
```

Sampai, di sini, cobalah untuk menjalankan kembali program. Tekanlah tombol untuk meja 01, kemudian kliklah button Tambah Pesanan 3 kali. Perhatikanlah gambar di halaman berikut.



Bisa kita lihat, setiap kali pengguna klik pada button Tambah Pesanan, pesanan Singkong baru akan ditambahkan (tampaknya, yang memesan adalah penyuka Singkong).

Cobalah tekan button meja 02. Kemudian, tekanlah button Tambah Pesanan 2 kali. Perhatikanlah bahwa table akan berisi 2 pesanan Singkong.

Lalu, kembalilah tekan button meja 01. Table akan tetap berisikan 3 baris. Cobalah juga untuk menekan button untuk meja-meja lain, dan buktikanlah bahwa kita telah dapat menambahkan pesanan. Walau, selalu berupa makanan Singkong.

Mari kita simpan kode aktual untuk penambahan pesanan (versi lain dari contoh-contoh sebelumnya) di contoh berikut. Untuk saat ini, berikut adalah kode program lengkap dalam 66 baris kode, yang semuanya telah kita sajikan sebelumnya:

Janganlah lupa menyesuaikan tulisan “meja_kosong” dan “meja_terisi” berikut dengan kode warna sesungguhnya, seperti green, ataupun yang didapatkan dari fungsi color_chooser.

```
var tables = {
  "01": [0, 0],
  "02": [1, 0],
  "03": [2, 0],
  "04": [3, 0],
  "05": [4, 0],
  "06": [0, 1],
  "07": [1, 1],
  "08": [2, 1],
  "09": [3, 1],
  "10": [4, 1],
  "11": [0, 2],
  "12": [1, 2],
  "13": [2, 2],
  "14": [3, 2],
  "15": [4, 2]
}

var table_status = {}
each(keys(tables), fn(e, i) {
  set(table_status, e, 0)
})

var button_color = {
```

```

    0: "meja_kosong",
    1: "meja_terisi"
}

var table_orders = {}
each(keys(tables), fn(e, i) {
    set(table_orders, e, [])
})

var table_active = []

reset()

var g = component("grid", "")
var g2 = component("grid", "")
var a = component("label", "Meja")
var t = component("table",
"Waktu,Kategori,Menu,Jumlah,Bungkus")
var ba = component("button", "Tambah Pesanan")
config(ba, "active", 0)
event(ba, fn() {
    var e = table_active[0]
    if (!empty(e)) {
        var ee = table_orders[e]
        var ee = ee + [@, "makanan", "Singkong", 1,
1]

        set(table_orders, e, ee)
    }
})

```

```

        config(t, "contents", table_orders[e])
    }
})
grid_add(g2, a, 0, 0, 1, 1, 0, 0, 1, 1)
grid_add(g2, t, 0, 1, 1, 1, 1, 1, 3, 1)
grid_add(g2, ba, 0, 2, 1, 1, 0, 0, 1, 1)
each(keys(tables), fn(e, i) {
    var v = tables[e]
    var b = component("button", e)
    grid_add(g, b, v[0], v[1], 1, 1, 1, 1, 3, 0)
    var s = table_status[e]
    var c = button_color[s]
    config(b, "background", c)
    event(b, fn () {
        config(a, "text", "Meja: " + e)
        config(t, "contents", table_orders[e])
        set(table_active, 0, e)
    })
})
add([g, g2])
show()

```

Halaman ini sengaja dikosongkan

Contoh 10: Bekerja dengan dialog

Di contoh 6, kita membuat frame dengan beberapa COMPONENT GUI, yang tujuan akhirnya adalah untuk menampilkan message box bahwa pesanan tertentu telah ditambahkan. Ini kemudian kita lengkapi pada contoh 7 dan 8, dengan tujuan akhir adalah menyimpan pesanan ke file CSV. Walau, sampai di sini, kita tidak tahu, pesanan-pesanan tersebut dari meja apa saja.

Di contoh 9, kita menampilkan button-button sesuai peletakan meja, menampilkan daftar pesanan setiap meja, dan menambahkan pesanan setiap tombol Tambah Pesanan ditekan. Hanya saja, pesanannya selalu berupa makanan singkong.

Idealnya, frame yang tadinya kita perbaiki sampai contoh 8 dapat digunakan kembali. Menambahkan ke table setiap meja sudah bisa. Menampilkan form tambah pesanan sudah bisa. Bagaimana kalau kita tampilkan ulang frame tersebut?

Sayangnya, kami harus mengecewakan Anda sampai di sini. Di bahasa Singkong, sebuah program GUI hanya dapat terdiri dari satu frame. Apa!? Anda mungkin terkejut dan sedikit kecewa.

Tujuannya adalah supaya program dengan GUI dapat dibuat dengan konsep yang sesederhana mungkin. Dengan demikian, fungsi show (atau hide, clear, closing, semua fungsi yang namanya diawali dengan frame, ataupun semua fungsi lain terkait frame) jelas merujuk ke satu-satunya frame per program.

Lalu, kan tidak mungkin juga kalau program yang dihasilkan dengan Singkong harus selalu menghapus ulang semua COMPONENT dari frame (dengan clear atau reset) dan menambahkan ulang semua COMPONENT lain ketika diperlukan. Tentu, pengembang bahasa Singkong tidak sampai setega itu (walau, sejujurnya, hanya inilah yang tersedia pada bahasa Singkong versi-versi awal).

Bagaimana kalau: kita jalankan program lain (dengan fungsi system), yang akan menampilkan framenya sendiri begitu diperlukan? Oh tidak!!! Lebih rumit, akan ada sejumlah keterbatasan secara teknis, mempersulit berbagi data antar COMPONENT, dan membutuhkan waktu lebih banyak (harus menjalankan interpreter Singkong lagi). Kita punya cara yang jauh lebih baik (dan tepat).

Yaitu: (A) dengan dialog atau (B) dengan popup (yang akan kita bahas pada buku lain). Dialog dapat terdiri dari panel atau grid, akan memiliki dekorasi window, dan harus secara ditutup eksplisit. Di sisi lain, popup bebas ditampilkan di mana saja, tidak memiliki dekorasi window, dapat tetap tampil, dan juga dapat terdiri dari panel atau grid.

Jadi, kedua pilihan dapat menggunakan panel atau grid. Di contoh ini, kita akan gunakan dialog dan grid. Mari kita sesuaikan program di contoh 6, dengan beberapa catatan berikut:

- Kita akan pertahankan kode untuk pembuatan dan layout COMPONENT. Ini berarti, setelah pemanggilan fungsi reset sampai add.
 - o Untuk dialog, kita tidak membutuhkan button Tambah, karena dialog akan terdiri dari tombol OK dan Cancel. Pesanan tidak diproses apabila dialog ditutup dengan Cancel, Escape keyboard, ataupun dengan tombol di title dialog.
- Kita akan sesuaikan event handler ketika button Tambah ditekan.
- Kita akan tuliskan ulang semua kode tersebut sebagai FUNCTION sendiri. Sebelum ini, kita sudah membuat FUNCTION ketika menggunakan fungsi each ataupun mendaftarkan event handler dengan fungsi event. Hanya saja, FUNCTION yang kita buat tidak diberikan nama. Kali ini, kita akan buat FUNCTION dengan nama: `add_order`.

Bukalah kembali kode program di contoh 9, dan tambahkan 41 baris kode berikut (yang diadaptasi dari contoh 6) di awal program, sebelum baris pertama (sebelum var tables).

```
var products = {
    "makanan": ["Bakwan", "Singkong", "Pangsit",
    "Perkedel"],
    "minuman": ["Air", "Kopi", "Teh"]
}

var add_order = fn(t) {
    var a1 = component("label", "Kategori")
    var a2 = component("label", "Menu")
    var a3 = component("label", "Jumlah")
    var c1 = component("combobox",
    ",Makanan,Minuman")
    var c2 = component("combobox", "")
    var s1 = component("spin", "1,1,20,1")
    var x1 = component("checkbox", "Bungkus")

    event(c1, fn() {
        var c = lower(get(c1, "text"))
        var p = products[c]
        if (p == null) {
            var p = []
        }
        config(c2, "contents", p)
    })
}
```

```

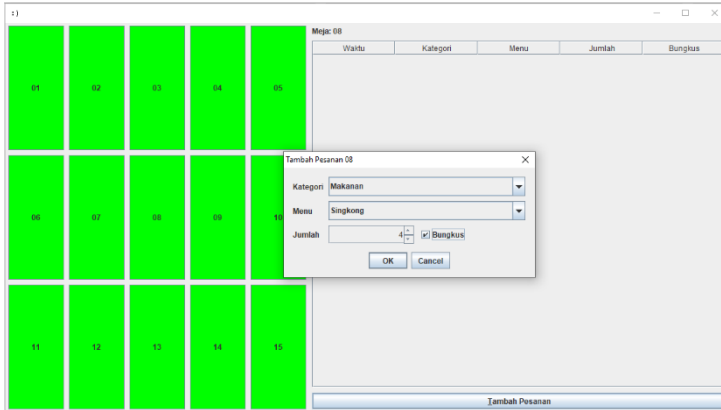
    })
    var g = component("grid", "")
    grid_add(g, a1, 0, 0, 1, 1, 0, 1, 1, 1)
    grid_add(g, c1, 1, 0, 2, 1, 1, 1, 3, 1)
    grid_add(g, a2, 0, 1, 1, 1, 0, 1, 1, 1)
    grid_add(g, c2, 1, 1, 2, 1, 1, 1, 3, 1)
    grid_add(g, a3, 0, 2, 1, 1, 0, 1, 1, 1)
    grid_add(g, s1, 1, 2, 1, 1, 1, 1, 3, 1)
    grid_add(g, x1, 2, 2, 1, 1, 1, 1, 1, 0)

    var r = panel_dialog(g, "Tambah Pesanan " + t,
400, 200)

    if (r == "OK") {
        var c = get(c2, "text")
        if (!empty(c)) {
            var k = get(c1, "text")
            var x = number_boolean(get(x1,
"active"))
            var s = get(s1, "contents")
            return [c, k, c, s, x]
        }
    }
    return null
}

```

Mari kita lihat, ketika button Tambah Pesanan ditekan, dialog akan ditampilkan. Menu bisa dipilih dan ketika tombol OK ditekan, maka pesanan akan ditambahkan ke table. Gambar berikut adalah pada meja 08:



Dan, ketika OK pada dialog ditekan, bisa kita lihat, selain pesanan berhasil ditambahkan ke table, button meja pun berubah menjadi warna ketika meja terisi. Seperti yang Anda mungkin telah duga, kode yang barusan ditambahkan tidak melakukan semua ini. Melainkan hanya menampilkan dialog.

Apabila kita lihat, kode yang baru ditambahkan tersebut, yang tidak diwarnai berbeda, pada dasarnya sama dengan contoh 6, dikurangi button Tambah.

Yang lebih menarik barangkali yang diwarnai biru. Sebagian mirip dengan event handler button Tambah. Yang berbeda, kita memanggil fungsi `panel_dialog` untuk menampilkan dialog dengan title dan ukuran tertentu. Fungsi ini mengembalikan `STRING` (misal "OK" ketika OK dipilih untuk menutup dialog).

Sementara, yang diwarnai hijau, adalah cara kita membuat `FUNCTION`, yang dalam contoh ini, membutuhkan satu argumen,

yaitu STRING nama meja (variabel `t`, yang lalu ditambahkan pada title dialog). FUNCTION akan mengembalikan ARRAY yang mirip dengan ARRAY pesanan makanan di contoh sebelumnya apabila valid, dan null selebihnya.

Bagaimana kita memanggil FUNCTION `add_order` ini? Mari kita lihat event handler ketika button Tambah Pesanan ditekan. Berikut adalah kode program yang telah disesuaikan:

```
event (ba, fn() {
    var e = table_active[0]
    if (!empty(e)) {
        var ee = table_orders[e]
        var oo = add_order(e)
        if (oo != null) {
            var ee = ee + oo
            set(table_orders, e, ee)
            config(t, "contents", table_orders[e])
        }
        if (!empty(ee)) {
            set(table_status, e, 1)
            var b = table_buttons[e]
            config(b, "background",
button_color[1])
        }
    }
})
```

Apabila kembalian FUNCTION `add_order` bukan null, kita tambahkan ARRAY pesanan baru ke pesanan sebelumnya, kemudian kita atur isi table (sebagian dari baris-baris yang diwarnai biru).

Lalu, apabila isi table bukanlah ARRAY kosong (kita cek dengan `empty`), maka sudah pasti kita perlu ubah status meja dan warnai sebagaimana disepakati sebelumnya (baris-baris kode yang diwarnai hijau).

Terakhir, apabila Anda mencermati, variable `table_buttons` belum ada sebelumnya. Ini adalah HASH, pemetaan dari nama meja ke button. Tambahkan baris berikut setelah `var table_active`:

```
var table_buttons = {}
```

Dan, baris berikut sebelum event handler setiap button meja:

```
set(table_buttons, e, b)
```

Kode program selengkapnya adalah sebagai berikut (117 baris):

Seperti sebelumnya, janganlah lupa menyesuaikan tulisan “`meja_kosong`” dan “`meja_terisi`” berikut dengan kode warna sesungguhnya, seperti `green`, ataupun yang didapatkan dari fungsi `color_chooser`. Atau, sebagai latihan tambahan, bagaimana kalau definisi warna ini disimpan dalam file konfigurasi? Apabila diperlukan, bacalah dokumentasi terkait fungsi-fungsi berikut: `properties_read` atau `properties_write`.

```
var products = {
  "makanan": ["Bakwan", "Singkong", "Pangsit",
    "Perkedel"],
  "minuman": ["Air", "Kopi", "Teh"]
}
```

```

var add_order = fn(t) {
    var a1 = component("label", "Kategori")
    var a2 = component("label", "Menu")
    var a3 = component("label", "Jumlah")
    var c1 = component("combobox",
",Makanan,Minuman")
    var c2 = component("combobox", "")
    var s1 = component("spin", "1,1,20,1")
    var x1 = component("checkbox", "Bungkus")

    event(c1, fn() {
        var c = lower(get(c1, "text"))
        var p = products[c]
        if (p == null) {
            var p = []
        }
        config(c2, "contents", p)
    })
    var g = component("grid", "")
    grid_add(g, a1, 0, 0, 1, 1, 0, 1, 1, 1)
    grid_add(g, c1, 1, 0, 2, 1, 1, 1, 3, 1)
    grid_add(g, a2, 0, 1, 1, 1, 0, 1, 1, 1)
    grid_add(g, c2, 1, 1, 2, 1, 1, 1, 3, 1)
    grid_add(g, a3, 0, 2, 1, 1, 0, 1, 1, 1)

```

```

grid_add(g, s1, 1, 2, 1, 1, 1, 1, 3, 1)
grid_add(g, x1, 2, 2, 1, 1, 1, 1, 1, 0)
var r = panel_dialog(g, "Tambah Pesanan " + t,
400, 200)
if (r == "OK") {
    var c = get(c2, "text")
    if (!empty(c)) {
        var k = get(c1, "text")
        var x = number_boolean(get(x1,
"active"))
        var s = get(s1, "contents")
        return [@, k, c, s, x]
    }
}
return null
}
var tables = {
    "01": [0, 0],
    "02": [1, 0],
    "03": [2, 0],
    "04": [3, 0],
    "05": [4, 0],
    "06": [0, 1],
    "07": [1, 1],

```

```

    "08": [2, 1],
    "09": [3, 1],
    "10": [4, 1],
    "11": [0, 2],
    "12": [1, 2],
    "13": [2, 2],
    "14": [3, 2],
    "15": [4, 2]
}

var table_status = {}
each(keys(tables), fn(e, i) {
    set(table_status, e, 0)
})
var button_color = {
    0: "meja_kosong",
    1: "meja_terisi"
}
var table_orders = {}
each(keys(tables), fn(e, i) {
    set(table_orders, e, [])
})
var table_active = [""]

```



```

var table_buttons = {}

reset()

var g = component("grid", "")
var g2 = component("grid", "")
var a = component("label", "Meja")
var t = component("table",
"Waktu,Kategori,Menu,Jumlah,Bungkus")
var ba = component("button", "Tambah Pesanan")
config(ba, "active", 0)
event(ba, fn() {
    var e = table_active[0]
    if (!empty(e)) {
        var ee = table_orders[e]
        var oo = add_order(e)
        if (oo != null) {
            var ee = ee + oo
            set(table_orders, e, ee)
            config(t, "contents", table_orders[e])
        }
        if (!empty(ee)) {
            set(table_status, e, 1)
            var b = table_buttons[e]

```

```

        config(b, "background",
button_color[1])
    }
}
})
grid_add(g2, a, 0, 0, 1, 1, 0, 0, 1, 1)
grid_add(g2, t, 0, 1, 1, 1, 1, 1, 3, 1)
grid_add(g2, ba, 0, 2, 1, 1, 0, 0, 1, 1)
each(keys(tables), fn(e, i) {
    var v = tables[e]
    var b = component("button", e)
    grid_add(g, b, v[0], v[1], 1, 1, 1, 1, 3, 0)
    var s = table_status[e]
    var c = button_color[s]
    config(b, "background", c)
    set(table_buttons, e, b)
    event(b, fn () {
        config(a, "text", "Meja: " + e)
        config(t, "contents", table_orders[e])
        set(table_active, 0, e)
    })
})
})
add([g, g2])
show()

```

Catatan: Apabila Anda perhatikan, kita tidak lagi menuliskan kode untuk menyimpan pesanan ke file tertentu. Bagaimana kalau ini dianggap sebagai soal latihan?

Anda mungkin menjawab: bukankah beberapa baris kodenya tinggal dicontoh dari ketika bekerja dengan file CSV?

Baiklah. Sebelum menuliskan kode programnya, kami berharap Anda berkenan mempertimbangkan beberapa hal berikut.

(A) Kita sudah dapat memisahkan pesanan per meja. Apakah pesanan dari semua meja akan tetap disimpan pada file yang sama? Ataukah, Anda ingin menggunakan file yang berbeda untuk setiap meja (barangkali, dengan nama file 'berdasarkan' nama meja)?

(B) Berdasarkan nama meja berarti ada kemungkinan tidak valid menurut sistem di mana penyimpanan dilakukan. Program yang Anda tulis akan dapat dijalankan di berbagai sistem operasi, dimana karakter tertentu mungkin tidak valid sebagai nama file. Atau, bahkan ketika sistem database relasional digunakan, karakter tertentu mungkin tidak bisa digunakan sebagai bagian dari nama tabel.

(C) Selain per meja, Anda mungkin ingin menyimpan pesanan per meja sebagai transaksi penjualan tersendiri. Setiap meja tentu ada kemungkinan ditempati oleh pelanggan yang berbeda, dalam waktu yang berbeda pula.

(D) Sampai di sini, kita mengasumsikan program dijalankan secara standalone. Jadi, hanya di komputer tersebut saja. Kenyataannya, di rumah makan, dapur, kasir, dan bagian lain mungkin menggunakan komputer masing-masing. Status per meja di contoh ini misalnya, tersimpan hanya pada memori sistem di mana program dijalankan. Agar komputer lain bisa mendapatkan 'status' ini, maka data ini harus bisa diakses bersama. Database? File sharing? HTTP Server? Anda tentukan :) Sebagiannya akan kita bahas dalam buku-buku lain.

Catatan: Kita berkali-kali menuliskan contoh program tertentu dituliskan dalam sekian baris. Lalu, apakah jumlah baris adalah satu-satunya cara dalam mengukur? Mari kita lihat dalam pengantar pengukuran piranti lunak berikut.

Pengukuran Piranti Lunak

Penulis: Benfano Soewito, Ph.D.

Jika kita ingin mengetahui mana yang lebih baik diantara dua buah laptop, maka kita akan membandingkan spesifikasinya. Pada umumnya kita akan membandingkan prosesor, memori (RAM), dan jenis hardisk-nya untuk menentukan laptop yang mana yang lebih baik. Artinya ada parameter-parameter yang jelas untuk membandingkan kedua laptop tersebut. Lalu bagaimana jika kita ingin membandingkan dua software yang mempunyai fungsi yang sejenis? Parameter-parameter apa yang harus digunakan?

Parameter yang sering digunakan untuk membandingkan software tersebut biasanya adalah jumlah baris dari program software tersebut. Tetapi pertanyaannya adalah apakah ini sudah benar? Karena pada sebuah program juga sering kali terdapat komentar atau catatan dari orang yang membuat program tersebut atau programmernya, kemudian beberapa baris dari program juga kadang kala dapat dijadikan menjadi satu baris. Sehingga parameter jumlah baris untuk menentukan software mana yang lebih baik menjadi kurang tepat.

Jika demikian, parameter apa yang dapat digunakan untuk mengukur sebuah software? Memang tidak mudah untuk melakukan pengukuran terhadap sebuah software, tetapi penelitian mengenai hal ini masih terus dilanjutkan. Sampai saat ini masih belum ada standar yang pasti untuk mengukur sebuah software. Tetapi beberapa cara untuk mengukur software telah diperkenalkan. Cara-cara ini dapat dibagi menjadi dua bagian, yaitu mengukur dan menghitung. Mengukur artinya langsung mengukur pada produk atau software tersebut, contohnya seperti parameter jumlah baris. Sedangkan menghitung dilakukan tidak secara langsung, parameter

yang diinginkan didapat dari hasil perhitungan, seperti menghitung kompleksitas software.

Pertanyaan selanjutnya, mengapa kita perlu mengukur sebuah software? Untuk menjawab pertanyaan ini kita harus kembali dulu kepada arti pengukuran. Pertama kita harus memahami bahwa tujuan dilakukan pengukuran adalah untuk membandingkan antara objek satu dengan yang lainnya. Jika kita tidak akan membandingkan maka tidak perlu dilakukan pengukuran. Kedua kita harus memahami arti pengukuran, pengukuran sebenarnya adalah memberi sebuah simbol kepada sebuah objek. Contohnya pengukuran tinggi sebuah pohon artinya kita memberikan label atau simbol meter atau centimeter kepada pohon tersebut untuk mengukur panjang.

Jika kita dapat mengukur software maka kita bisa menjawab beberapa pertanyaan berikut:

- Berapa banyak perbaikan yang sudah dilaksanakan dalam pengembangan sebuah software?
- Berapa ongkos untuk membuat sebuah software?
- Bagaimana trend ukuran besar software di masa mendatang?
- Seberapa produktif seorang programmer?
- Seberapa kompleks sebuah software?
- Kapan sebuah software akan tidak dapat berfungsi normal?

Mengukur sebuah objek harus didasarkan pada tujuannya, artinya kita mau membandingkan apa pada objek tersebut. Contohnya seperti mengukur orang, jika kita ingin mengetahui kesehatan seseorang maka dilakukan pengecekan darah, jika kita ingin mengetahui tinggi dan berat, maka dilakukan pengukuran dengan meteran dan timbangan. Demikian juga dengan software. Mengukur sebuah software harus didasari dengan tujuan juga. Sebelum melakukan pengukuran pada sebuah software kita harus menentukan dulu apa yang mau kita bandingkan.

Untuk mengukur software pada umumnya dapat dibagi menjadi tiga kategori, yaitu pengukuran proses, pengukuran produk dan pengukuran resources (sumber daya). Mari kita lihat satu persatu apa yang dimaksud dari tiga kategori tersebut.

Pengukuran proses dilakukan jika kita ingin mengetahui keadaan atau aktifitas pada saat proses pembuatan software tersebut. Seringkali yang diukur pada saat proses ini adalah variable waktu, seperti berapa lama sebuah fungsi dapat diselesaikan oleh programmer A yang sudah mempunyai pengalaman lebih dari lima tahun dibandingkan dengan programmer B yang baru lulus dari universitas. Parameter pengukuran dalam proses sangat berpengaruh untuk menentukan harga atau biaya pembuatan untuk sebuah software.

Selain parameter waktu, parameter jumlah error atau bug, yang dalam bahasa inggris dikenal dengan “number of faults”, juga menjadi sangat penting dalam pengukuran pada saat proses pembuatan sebuah software. Jumlah error ini dapat digunakan sebagai acuan atau pengukuran seberapa sulit pengerjaan dari sebuah software dan juga sebagai pengukuran effort dan kemampuan, keahlian seorang programmer.

Pengukuran jumlah error ini didapatkan pada saat testing, baik pada saat testing sebagian atau testing function dan juga pada saat testing software secara keseluruhan. Jika jumlah error terlalu banyak pada saat testing maka pengerjaan software secara keseluruhan akan lebih memakan waktu karena harus dilakukan perbaikan perbaikan dan ini tentunya akan mengakibatkan biaya pembuatan software menjadi lebih tinggi.

Pengukuran produk dilakukan pada produk atau program dari software itu sendiri. Pengukuran pada produk dapat dilakukan bukan hanya pada produk jadi atau software secara keseluruhan, tetapi juga

dapat dilakukan pengukuran pada module, function atau bagian dari software itu sendiri. Atribut untuk pengukuran product biasanya adalah size, complexity, modularity, usability, efficiency, reliability dan sebagainya.

Untuk pengukuran size, parameternya bisa line of code tanpa komentar atau bisa juga size dari executable filenya. Kadang kadang juga diukur berapa besar memory yang dibutuhkan untuk menjalankan software ini. Kemudian untuk mengukur complexity dari sebuah software, kita dapat menghitung berapa banyak cabang atau nested dari software tersebut. Nested atau cabang ini didapat dengan menghitung berapa banyak statement if-then, if-else, while, for, yang digunakan dalam sebuah software. Semakin banyak jumlah statement if-then, if-else, while dan for, yang digunakan pada sebuah software maka semakin kompleks software tersebut.

Pengukuran resources adalah pengukuran semua support yang digunakan dalam proses pembuatan software, baik manusia atau man power, hardware dan tools. Contoh atribut untuk pengukuran pada man power adalah pengalaman, usia, produktifitas, dan sebagainya. Pengalaman dan usia dapat langsung diukur dengan parameter waktu yaitu tahun, tetapi untuk produktifitas tidak dapat diukur langsung, jadi produktifitas harus dihitung. Oleh karena itu produktifitas disebut external atribut karena tergantung dari hal hal yang lainnya. Produktifitas sangat tergantung dari pengalaman programmer, jenis software yang dikembangkan, re-use, dan hardware serta tools yang digunakan oleh programmer. Dalam hal ini yang dimaksud software adalah komputer, monitor, mikrofon, kamera dan sebagainya. Produktifitas seharusnya akan meningkat seiring dengan meningkatnya spesifikasi dari komputer yang digunakan untuk membuat software. Begitu juga tools, dalam hal ini yang dimaksud tools adalah semua aplikasi yang digunakan untuk membuat software termasuk operating system-nya.

Ada kalanya sebuah software dapat menggunakan ulang statement atau function, yang disebut re-use. Semakin banyak penggunaan re-use ini maka produktifitas akan meningkat juga.

Terakhir, kita juga harus mempunyai catatan atau dokumentasi yang baik mengenai perubahan, penambahan opsi, ataupun perbaikan pada sebuah software. Catatan ini biasanya disebut track changes atau evolution dan lebih dikenal dengan versi.

Catatan harus meliputi,

1. Alasan mengapa dilakukan perbaikan atau perubahan.
2. Bagian mana saja yang diubah.
3. Siapa yang melakukan perubahan.
4. Kapan atau tanggal perubahan.
5. Hasil testing.
6. Nomor versi.

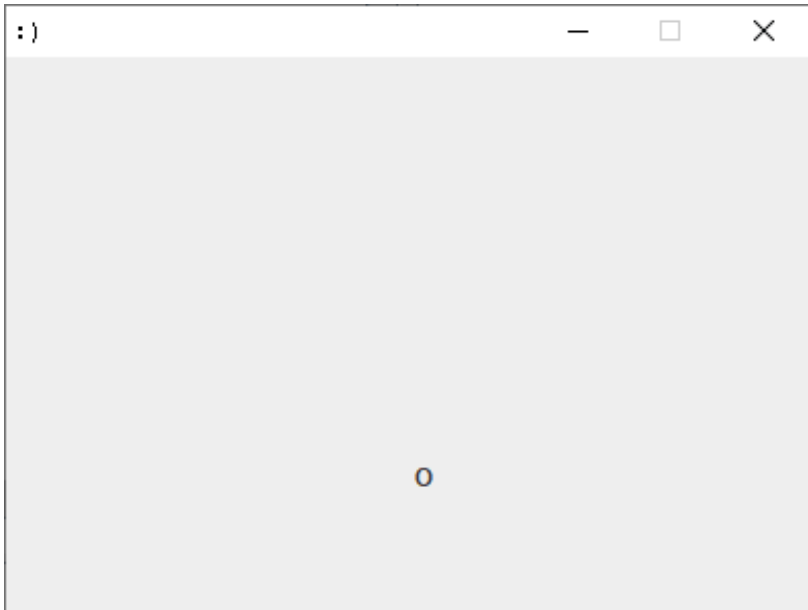
Oleh karena itu, salah satu kegunaan pengukuran pada software adalah untuk menganalisa hasil testing (point 5 diatas). Jika tidak ada pengukuran kita tidak tahu apakah hasil testing setelah dilakukan perubahan pada software tersebut lebih baik atau tidak. Untuk itu harus ditentukan parameter apa saja yang akan dijadikan pembanding sebelum dilakukan perubahan dan setelah dilakukan perubahan.

Dengan demikian dapat diambil kesimpulan bahwa aktivitas pengukuran dan analisis memungkinkan kita untuk:

- Menentukan karakteristik atau mendapatkan pemahaman tentang proses, produk, sumber daya, dan lingkungan dari sebuah software.
- Untuk dapat mengevaluasi, dan untuk menentukan status software yang sedang dibuat sehubungan dengan rencana kita.

- Dapat memprediksi kualitas dari software, dengan memahami hubungan antara proses dan produk. Selain itu, parameter-parameter yang kita ukur untuk beberapa atribut dapat digunakan untuk memprediksi hal-hal selain kualitas.
- Dapat digunakan untuk meningkatkan mutu produk, kesalahan proses dan menentukan resources yang tepat, dengan mengidentifikasi hambatan, akar penyebab, inefisiensi, dan peluang lain untuk perbaikan.

Bonus: Bola pantul



Anggaplah terdapat sebuah COMPONENT yang akan bergerak secara konstan di frame. Misal, setiap 100 milidetik, COMPONENT tersebut akan bergerak ke bawah dan kanan. Ketika mencapai batas bawah frame, maka COMPONENT tersebut dipantulkan sehingga bergerak ke atas dan kanan. Namun, sesampainya di sisi kanan frame, maka kembali dipantulkan sehingga arahnya menjadi ke atas dan kiri. Apa yang terjadi ketika mencapai sisi atas frame? Tentunya arahnya kembali dipantulkan menjadi ke bawah dan kiri. Demikian seterusnya.

Pada saat buku ini ditulis, terdapat 21 tipe COMPONENT, sebagai berikut:

```
> components ()
```

```
["barchart", "button", "checkbox", "combobox",  
"date", "draw", "edit", "grid", "image", "label",  
"mask", "panel", "password", "piechart", "progress",  
"radio", "spin", "tab", "table", "text", "view"]
```

```
> len(components())
```

```
21
```

Jadi, kabar gembiranya, Anda dapat menggunakan hampir semua dari COMPONENT tersebut untuk dipantul-pantulkan. Dengan COMPONENT draw, kita bahkan bisa menggambar sendiri selayaknya di kanvas. Atau, dengan COMPONENT image, dimana kita bisa membaca dari file gambar.

Anda mungkin bertanya. Satu pertanyaan mendasar yang sangat penting. Hanya pada contoh 4, kita menempatkan COMPONENT pada posisi tertentu pada panel. Dengan prinsip serupa, kita bisa menempatkan tipe COMPONENT lain. Atau, dengan COMPONENT draw yang disebut sebelumnya, kita bisa menggambar (dan menghapus) pada area tertentu. Tapi, bagaimana kita melakukan sesuatu secara berkala, misal setiap detik?

Anda mungkin menjawab pertanyaan Anda sendiri dengan mencoba melihat daftar fungsi bawaan, dengan melihat keluaran dari pemanggilan fungsi builtins. Apakah fungsi delay? Sayangnya tidak. Fungsi delay digunakan untuk menunda dan mungkin berdampak tertentu pada GUI.

Dari ratusan fungsi bawaan, Anda mungkin akhirnya menemukan timer, yang dapat memanggil FUNCTION tertentu setiap jeda sekian milidetik. Selamat kepada Anda, karena itulah jawabannya untuk bola (atau COMPONENT) pantul ini.

Sekarang, apa yang akan kita lakukan dalam FUNCTION tersebut? Cara termudah adalah dengan memposisikan COMPONENT tertentu secara absolut pada panel. Mari kita lihat potongan kode program berikut:

```
timer(100, fn() {  
    var x = d[0]  
    var y = d[1]  
    var dx = d[2]  
    var dy = d[3]  
    var xs = x + s  
    var ys = y + s  
    if (xs >= w | xs < s) {  
        var dx = dx * -1  
    }  
    if (ys >= h | ys < s) {  
        var dy = dy * -1  
    }  
    var x = x + dx  
    var y = y + dy  
    set(d, 0, x)  
    set(d, 1, y)  
    set(d, 2, dx)  
    set(d, 3, dy)  
    panel_add(p, b, x, y, s, s)
```

})

Dengan memanggil fungsi setiap 100 milidetik tersebut, kita mencapai efek label pantul dengan cara:

- Mendapatkan posisi x , y , jumlah piksel ketika berpindah secara horisontal (dx), jumlah piksel ketika berpindah secara vertikal (dy).
- Menjumlahkan posisi x dan y dengan ukuran label yang ingin ditambahkan (xs , ys)
- Jika: posisi x label \geq lebar frame atau $<$ lebar label, maka dx dikalikan -1 .
- Jika: posisi y label \geq tinggi frame atau $<$ tinggi label, maka dy dikalikan -1 .
- Menjumlahkan x dengan dx .
- Menjumlahkan y dengan dy .
- Menambahkan ke panel sesuai posisi dan ukuran label.

Kedengarannya merepotkan (terutama langkah terakhir), namun setidaknya bekerja, walau tidak mulus.

Berikut adalah baris kode tambahan, sebelum fungsi timer dipanggil tersebut:

```
reset ()  
  
var w = 400  
var h = 300  
var d = [w/2, h/2, 10, 10]  
var s = 30  
var b = component("label", "O")  
var p = component("panel", "")  
add(p)
```

```
size(w, h)
resizable(false)
show()
```

Bisa kita lihat bersama, kita telah menentukan ukuran frame dan memastikan tidak dapat diubah. Apabila tidak, dan pengguna mengubah ukuran frame, maka kita perlu menangani event frame dan selalu mendapatkan ukuran aktualnya.

Mari coba kita perbaiki contoh ini di buku berikutnya :)

Halaman ini sengaja dikosongkan

Daftar Pustaka

Noprianto. (2020). Mengenal dan Menggunakan Bahasa Pemrograman Singkong. PT. Stabil Standar Sinergi.

Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach. McGraw-Hill Higher Education.

Buku ini berisi sejumlah contoh source code dan penjelasan langkah demi langkah yang mudah dipahami untuk membuat program komputer yang dilengkapi Graphical User Interface (GUI), dengan bahasa pemrograman Singkong.

Anda tidak perlu memahami bahasa Singkong terlebih dahulu untuk mengikuti pembahasan dalam buku ini. Bisa menggunakan komputer dan dapat mengikuti langkah instalasi program pun sudah cukup. Pernah membuat program sebulan atau 30 tahun lalu? Itu lebih dari cukup.

Melengkapi contoh program, buku ini juga membahas pengantar untuk rekayasa dan pengukuran piranti lunak.



Dr. Noprianto mengembangkan bahasa pemrograman Singkong dan interpretasinya sejak akhir 2019.

Beliau menyukai pemrograman, dan mendirikan serta mengelola perusahaan pengembangan software dan teknologi Singkong.dev (PT. Stabil Standar Sinergi).

Noprianto menyelesaikan pendidikan doktor ilmu komputer dan telah menulis beberapa buku pemrograman (termasuk Python, Java, dan Singkong).

Buku dan softwarena dapat didownload dari <https://nopri.github.io>



Dr. Karto Iskandar, S.Kom., MM adalah Dosen di Fakultas Ilmu Komputer Universitas Bina Nusantara.

Tentunya mengajar, meneliti, melakukan publikasi, dan melayani masyarakat adalah kegiatan rutinnnya. Beliau juga bekerja sebagai Manajer di Knowledge Innovation.

Berbagai penghargaan lokal dan global terkait Knowledge Management telah diraihny dalam 10 tahun terakhir. Selain itu, berbagai penelitian lokal dan internasional telah beliau lakukan, termasuk menerbitkan beberapa artikel ilmiah di jurnal internasional yang terindeks Scopus.

Rekayasa Perangkat Lunak, Sistem Manajemen Pengetahuan, Mobile Application, E-Apps adalah beberapa bidang penelitiannya.



Benfano Soewito, Ph.D. adalah seorang Dosen di Universitas Bina Nusantara, di mana beliau mengajar dan melakukan penelitian di bidang jaringan, keamanan internet, dan internet of things.

Benfano telah bekerja sebagai peneliti di bidang TI sejak sedang menyelesaikan pendidikan master dan doktor dengan fokus pada jaringan dan keamanan sistem internet. Benfano memiliki lebih dari dua puluh tahun pengalaman sebagai peneliti dan telah menerbitkan lebih dari 160 makalah pada konferensi dan jurnal yang bereputasi.

Benfano juga telah bekerja di industri sebagai spesialis TI untuk memecahkan masalah perangkat lunak dan perangkat keras yang terkait dengan infrastruktur internet untuk memastikan sistem beroperasi dengan baik dan aman.

singkong.dev

Alamat: Puri Indah Financial Tower
Lantai 6, Unit 0612
Jl. Puri Lingkar Dalam Blok T8
Kembangan, Jakarta Barat 11610
Email: info@singkong.dev

ISBN 978-602-52770-3-0



Rp. 50.000