

DB Programming - Cont

Database Systems

Agenda

- × Project Details
- × SWT
- × Updating UI (Why do we need Threads?)

Agenda

- × Project Details
- × SWT
- × Updating UI (Why do we need Threads?)

Other JAVA UI

- × AWT (Abstract Windowing Toolkit)
 - standard for all platforms
 - too simple..
 - “Least Common Denominator”
- × SWING
 - try to fix AWT’s problems
 - uses AWT
 - complicated to use and learn
 - looks the same on every platform

View Sales Order

Order number: 0000006973

Document date: 24-04-2002

Purchase order number: DG-19970626-300

Requested delivery date: 06-05-2002

Delivery block: ☐

Order items:

Item	Material	Description	Quantity	Net value	Curr
000010	P-100	Pumpe PRECISION 100	1.000	2607.60	EUR
000020	P-100	Pumpe PRECISION 100	2.000	5215.20	EUR

Return info from BAPI:

CHARVA Example

File Options Help

Proxy Cookies Accounts

New User

Account name: leonid

Full name: Leonid Brezhnev

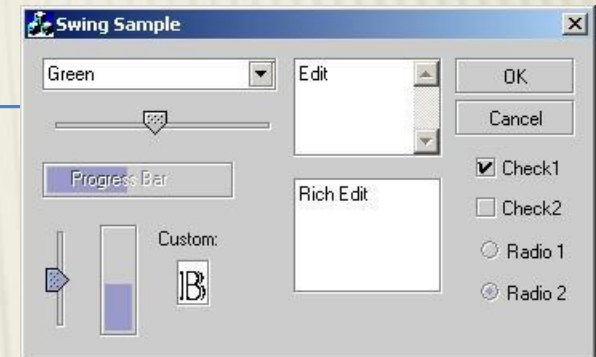
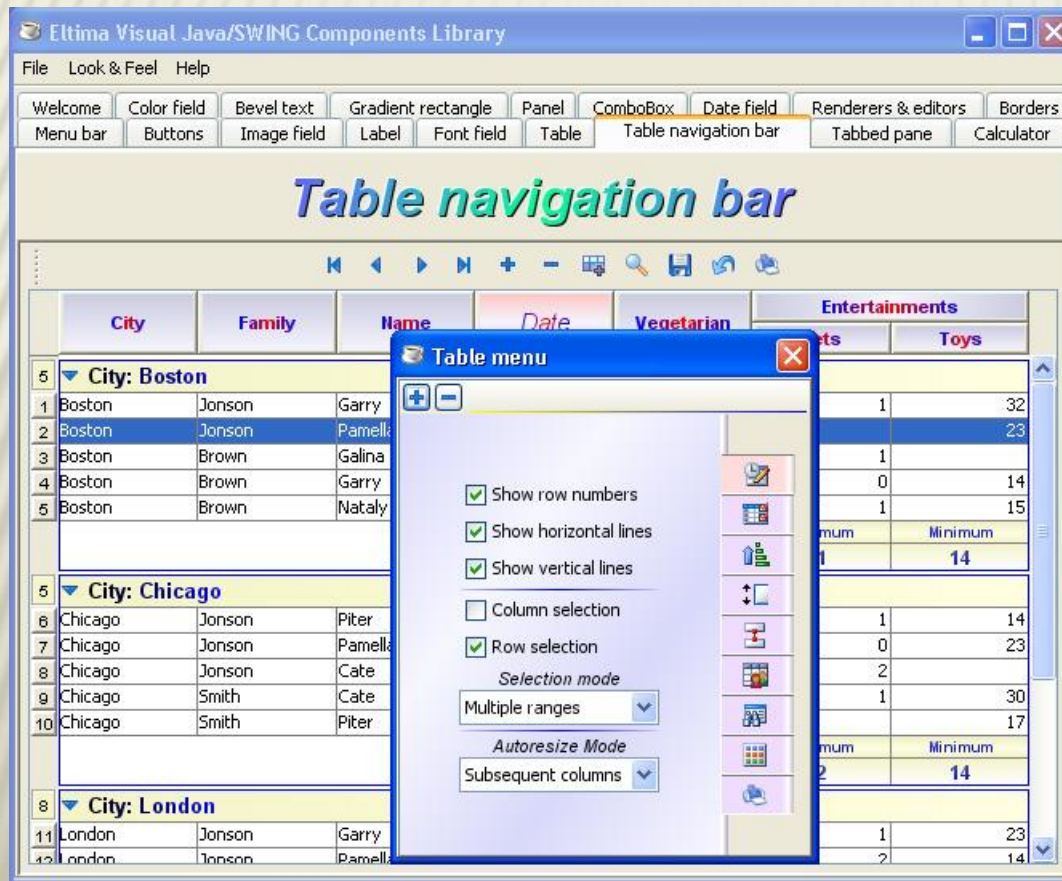
Password: *****

Repeat Password: *****

User accounts

User	Full Name
karl	Karl Marx
vlad	Vladimir Lenin

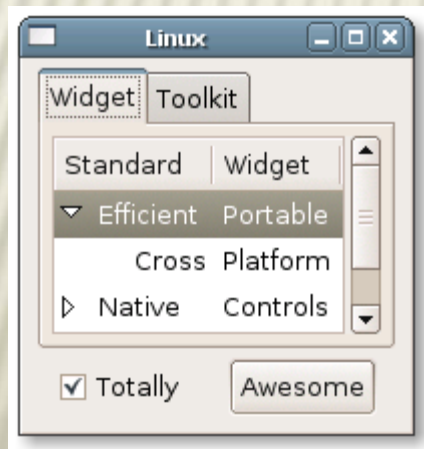
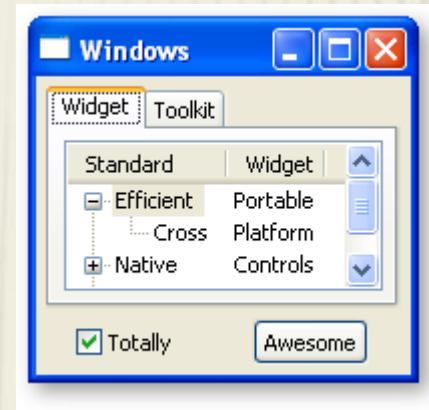
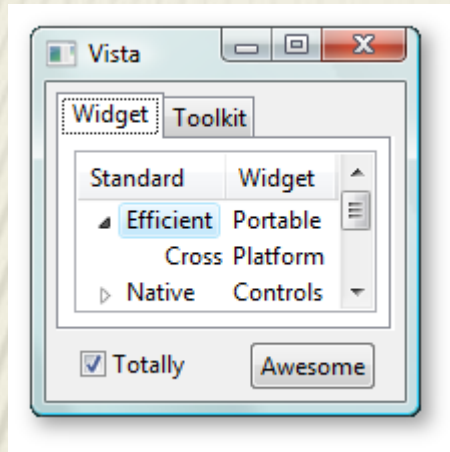
Swing



SWT (Standard Widget Toolkit)

- × Developed by IBM, maintained today by Eclipse
- × Easy implementation
- × Not portable – requires implementation for each platform. BUT, all major ones has 😊
 - “LCD” fixed 😊
- × Had performance issues, but today its fixed 😊

SWT - Takes the look of the OS



“Installing” SWT

- × Same as always:

- Add the right swt.jar (windows/unix/osx)

- (<http://www.eclipse.org/swt/>)

- import org.eclipse.swt.widgets.*

- × (same as “installing” JDBC)

Widgets

- ✖ Widget is the “UI element”
(window, button, icon...)
- ✖ When creating a Widget, we need to supply its parent
- ✖ Every Widget needs to be disposed when done.
→ Luckily, disposing the parents disposes of its Childs

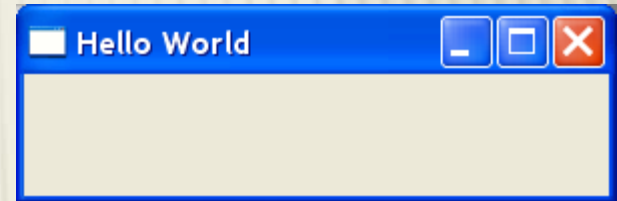
Hello World

```
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.widgets.Shell;
public class HelloWorldAlone
{
    public static void main(String[] args)
    {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("Hello World");
        shell.setSize(300, 100);

        shell.open();

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

        display.dispose();
    }
}
```



Hello World – A few more words

- ✖ Shell is the main window. As any widget, he must have a parent:
→ Display (windows..)
- ✖ If you wont keep the Shell open (listening to events) the program will immediately terminate
- ✖ Can we have a more “swing” environment?

SWTUtil

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

public class SWTUtil{
    private static Display display = new Display();

    public static Shell getShell(){
        Shell shell = new Shell(display);
        return shell;
    }

    public static void openShell(Shell shell) {
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }

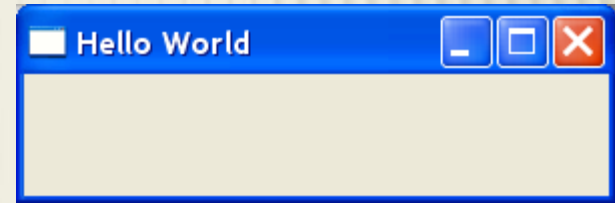
        display.dispose();
    }
}
```


Hello World (with SWTUtil)

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

public class HelloWorld
{
    public static void main(String[] args)
    {
        Shell shell = SWTUtil.getShell();
        shell.setText("Hello World");
        shell.setSize(300, 100);

        SWTUtil.openShell(shell);
    }
}
```



More on Widget

- × Widgets are created by:
 - Specifying parent
 - Specifying style
- × A parent is the container that the widget is created inside (e.g. Shell)
- × A style is a constant from the SWT class (SWT.BORDER, SWT.LEFT, SWT.NONE ...)
- × Multiple styles can be joined with “|”
SWT.V_SCROLL|SWT.H_SCROLL| SWT.BORDER

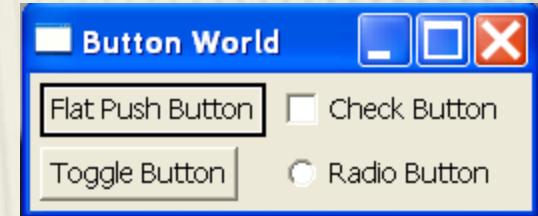
Label

```
Shell shell = SWTUtil.getShell();  
shell.setText("Label World");  
shell.setLayout(new GridLayout()); // layouts are explained later
```



```
// Create labels  
new Label(shell, SWT.NONE).setText("Regular label");  
new Label(shell, SWT.SEPARATOR);  
new Label(shell, SWT.SEPARATOR|SWT.HORIZONTAL);  
  
// pack and show  
shell.pack();  
SWTUtil.openShell(shell);
```

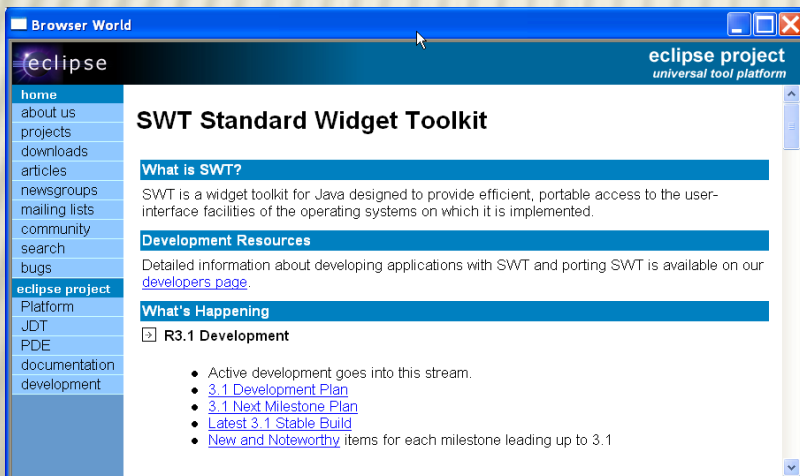
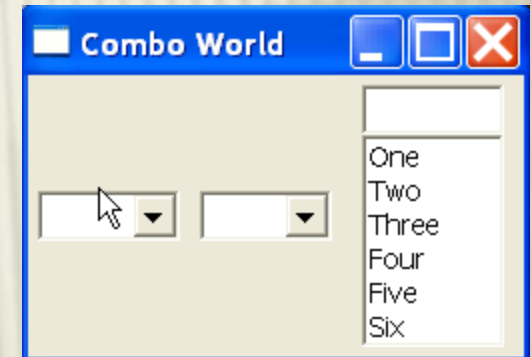
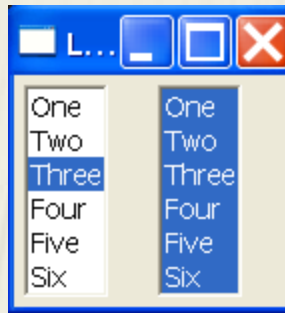
Button



```
shell.setText("Button World");  
shell.setLayout(new GridLayout(2, true)); // layouts are explained later  
  
new Button(shell, SWT.PUSH | SWT.FLAT).setText("Flat Push Button");  
new Button(shell, SWT.CHECK).setText("Check Button");  
new Button(shell, SWT.TOGGLE).setText("Toggle Button");  
new Button(shell, SWT.RADIO).setText("Radio Button");
```

Some more Widgets

- ✧ Take a look at the SWT Tutorial PPT (on the course slides page)



Who's your daddy??

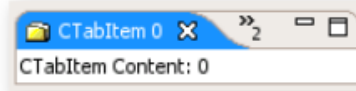
Some more Widgets (2)

- × <http://www.eclipse.org/swt/widgets/>



CoolBar

[javadoc](#) - [snippets](#)



CTabFolder

[javadoc](#) - [snippets](#)



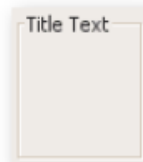
DateTime

[javadoc](#) - [snippets](#)



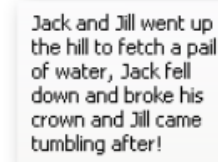
ExpandBar

[javadoc](#) - [snippets](#)



Group

[javadoc](#)

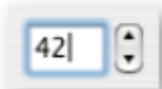


Label

[javadoc](#) - [snippets](#)

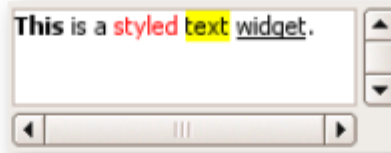
Some more Widgets (3)

- × <http://www.eclipse.org/swt/widgets/>



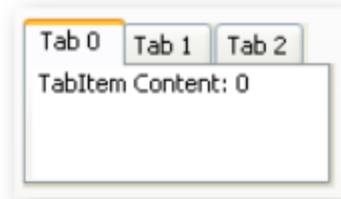
Spinner

[javadoc](#) - [snippets](#)



StyledText

[javadoc](#) - [snippets](#)



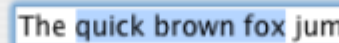
TabFolder

[javadoc](#) - [snippets](#)

Name	Type	Size
☐ Index:0	classes	0
☒ Index:1	databases	2556
☐ Index:2	images	9157
☒ Index:3	classes	0
☐ Index:4	databases	2556

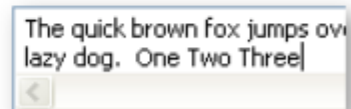
Table

[javadoc](#) - [snippets](#)



Text (SWT.SINGLE)

[javadoc](#) - [snippets](#)



Text (SWT.MULTI)

[javadoc](#) - [snippets](#)

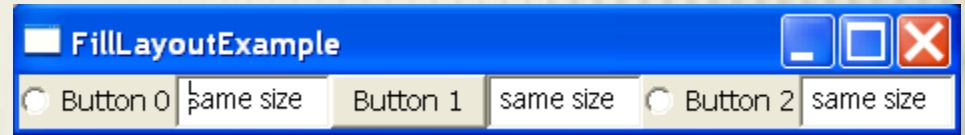
Layouts

- × First introduced in AWT
- × Ease burden of laying out components
- × SWT offers 5 layouts:
 - FillLayout
 - RowLayout
 - GridLayout
 - FormLayout
 - (*) Stack Layout
- × <http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html>

FillLayout

- × Places all widgets in either a single column or row (SWT.VERTICAL,SWT.HORIZONTAL)
- × Makes all widgets the same size

FillLayout



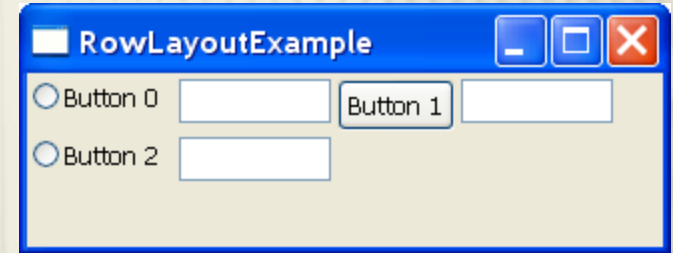
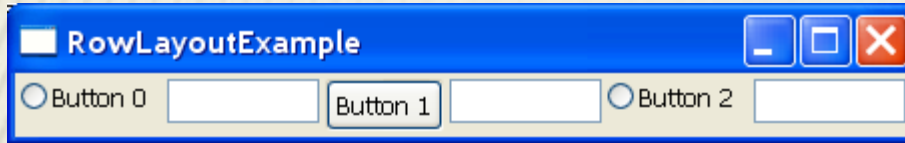
```
shell.setLayout(new FillLayout(SWT.HORIZONTAL));
```

```
for(int i = 0; i < 3; i ++)  
{  
    new Button(shell,  
        (i % 2 == 0) ? SWT.RADIO : SWT.PUSH).setText("Button " + i);  
    new Text(shell, SWT.BORDER).setText("same size");  
}
```


RowLayout

- ✖ Places all widgets in either a single column or row (SWT.VERTICAL,SWT.HORIZONTAL)
- ✖ Doesn't force all widgets to be the same size
- ✖ Can wrap to a new row or column if it runs out of space
- ✖ Can use **RowData** objects to determine initial heights/widths for controls

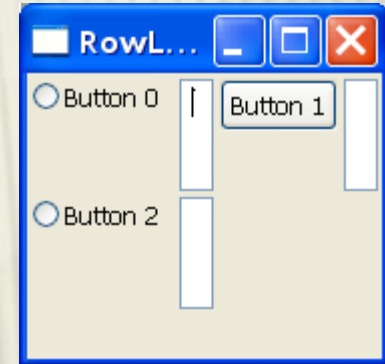
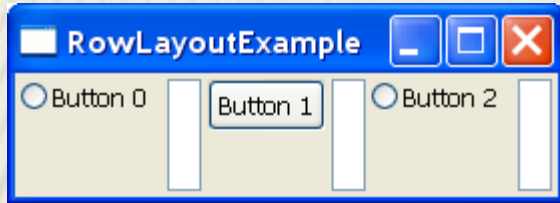
RowLayout



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

```
for(int i = 0; i < 3; i++) {  
    new Button(shell,  
        (i % 2 == 0) ? SWT.RADIO : SWT.PUSH).setText("Button " + i);  
    new Text(shell, SWT.BORDER);  
}
```

RowLayout

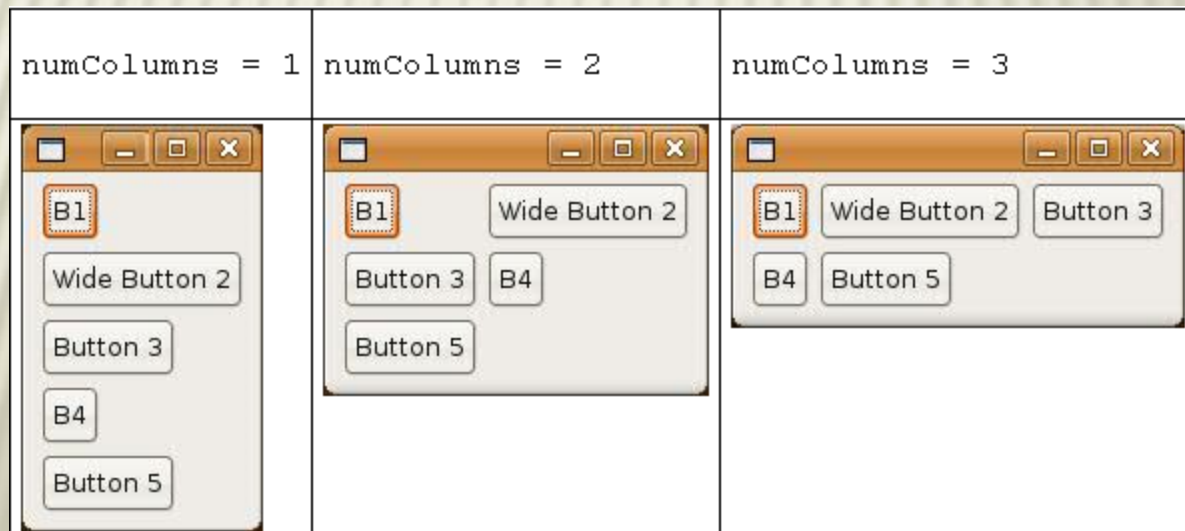


```
shell.setLayout(new RowLayout(SWT.HORIZONTAL));
```

```
for(int i = 0; i < 3; i ++) {  
    new Button(shell,  
        (i % 2 == 0) ? SWT.RADIO : SWT.PUSH).setText("Button " + i);  
    new Text(shell, SWT.BORDER).setLayoutData(new RowData(5, 50));  
}
```

GridLayout

- ✖ Lays out controls in a grid
- ✖ Added from left to right, new row is created when `numColumns + 1` Widgets are added



GridLayout – Main Properties

- ✖ `int horizontalSpacing` – horizontal space in pixels between adjacent cells
- ✖ `int verticalSpacing` – vertical space in pixels between adjacent cells
- ✖ `boolean makeColumnsEqualWidth` – forces all columns to be same width
- ✖ `int marginWidth` – margin in pixels along right and left edges
- ✖ `int marginHeight` – margin in pixels along top and bottom edges
- ✖ `int numColumns` – number of columns for the layout

GridData

- × Provide better control..
`widget.setLayoutData(GridData)`
- × Lots of options...check the API
- × **Warning** for Swing programmers – DO NOT TRY TO REUSE GridData objects (simply create new for each widget)

GridData - Example

<pre>horizontalAlignment = GridData.BEGINNING</pre> (default)	 A GUI window with a title bar and three buttons: 'Wide Button 2', 'Button 3', and 'Button 5'. The buttons are arranged in a grid. 'Wide Button 2' and 'Button 3' are in the top row, and 'Button 5' is in the bottom row. The 'Wide Button 2' button is highlighted with a red border.
<pre>horizontalAlignment = GridData.CENTER</pre>	 A GUI window with a title bar and three buttons: 'Wide Button 2', 'Button 3', and 'Button 5'. The buttons are arranged in a grid. 'Wide Button 2' and 'Button 3' are in the top row, and 'Button 5' is in the bottom row. The 'Wide Button 2' button is highlighted with a red border.
<pre>horizontalAlignment = GridData.END</pre>	 A GUI window with a title bar and three buttons: 'Wide Button 2', 'Button 3', and 'Button 5'. The buttons are arranged in a grid. 'Wide Button 2' and 'Button 3' are in the top row, and 'Button 5' is in the bottom row. The 'Wide Button 2' button is highlighted with a red border.
<pre>horizontalAlignment = GridData.FILL</pre>	 A GUI window with a title bar and three buttons: 'Wide Button 2', 'Button 3', and 'Button 5'. The buttons are arranged in a grid. 'Wide Button 2' and 'Button 3' are in the top row, and 'Button 5' is in the bottom row. The 'Wide Button 2' button is highlighted with a red border.

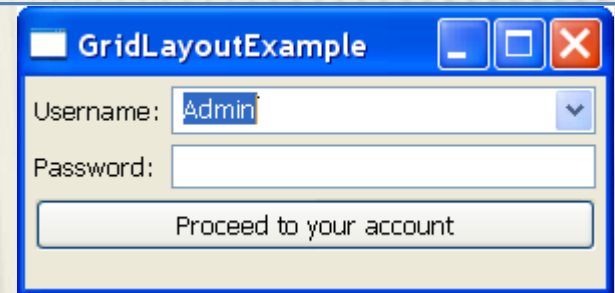
Another Example

```
shell.setLayout(new GridLayout(2, false));
```

```
new Label(shell, SWT.NONE).setText("Username:");  
Combo cmbUsername = new Combo(shell, SWT.DROP_DOWN);  
cmbUsername.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));  
cmbUsername.setItems(new String[]{"Howard", "Admin", "Kalman"});  
cmbUsername.setText("Admin");
```

```
new Label(shell, SWT.NONE).setText("Password:");  
new Text(shell, SWT.BORDER | SWT.PASSWORD).setLayoutData(new  
    GridData(GridData.FILL_HORIZONTAL));
```

```
Button loginButton = new Button(shell, SWT.PUSH | SWT.FLAT);  
loginButton.setText("Proceed to your account");  
GridData data = new GridData(GridData.FILL_HORIZONTAL);  
data.horizontalSpan = 2; // span 2 columns  
loginButton.setLayoutData(data);
```



Another Example

```
GridData gridData = new GridData();  
gridData.horizontalAlignment = GridData.FILL;  
gridData.horizontalSpan = 2;  
button5.setLayoutData(gridData);
```



```
GridData gridData = new GridData();  
gridData.horizontalAlignment = GridData.FILL;  
gridData.horizontalSpan = 2;  
button2.setLayoutData(gridData);
```



```
GridData gridData = new GridData();  
gridData.verticalAlignment = GridData.FILL;  
gridData.verticalSpan = 2;  
button3.setLayoutData(gridData);
```



Google for other examples



A screenshot of a simple graphical user interface window. It has a title bar with standard window controls. The form is divided into three sections. The first section is labeled 'Name:' and contains a single-line text field with the text 'Text grows horizontally'. The second section is labeled 'Address:' and contains a multi-line text area with the text 'This text field and the List below share any excess space.'. The third section is labeled 'Sports played:' and contains a list box with two items: 'Hockey' and 'Street Hockey'.

Name: Text grows horizontally

Address: This text field and the List below share any excess space.

Sports played:

- Hockey
- Street Hockey



A screenshot of a 'Dog Show Entry' form window. It has a title bar with standard window controls. The form is divided into several sections. The first section is labeled 'Dog's Name:' and contains a single-line text field with the text 'Bifford'. The second section is labeled 'Breed:' and contains a dropdown menu with the text 'Black Lab'. The third section is labeled 'Photo:' and contains a multi-line text area with a 'Browse...' button and a 'Delete' button. The fourth section is labeled 'Categories' and contains a list box with the following items: 'Best of Breed', 'Prettiest Female', 'Handsome Male', 'Best Dressed', 'Fluffiest Ears', 'Most Colors', 'Best Performer', 'Loudest Bark', 'Best Behaved', 'Prettiest Eyes', and 'Most Hair'. The fifth section is labeled 'Owner Info' and contains two single-line text fields: 'Name:' with the text 'Mary Smith' and 'Phone:' with the text '123-4567'. There is an 'Enter' button at the bottom right.

Dog Show Entry

Dog's Name: Bifford

Breed: Black Lab

Photo:

Browse...

Delete

Categories

- Best of Breed
- Prettiest Female
- Handsome Male
- Best Dressed
- Fluffiest Ears
- Most Colors
- Best Performer
- Loudest Bark
- Best Behaved
- Prettiest Eyes
- Most Hair

Owner Info

Name: Mary Smith

Phone: 123-4567

Enter

× <http://www.eclipse.org/articles/article.php?file=Article-Understanding-Layouts/index.html>

FormLayout

- ✖ Considered the most complex layout of SWT
- ✖ Based on $y = ax + b$ (not that most people who use it care)
- ✖ MAXIMUM flexibility
- ✖ People who understand it – love it 😊
- ✖ Needs a tutorial of its own and is therefore not covered here ...

Event Handling

- × Similar to Swing..
- × **Listener** is basically an interface that defines when certain behaviors happen
- × **Listeners** are attached to widgets
- × **Adapters** implements the interfaces

Popular Listeners / Adapters

- ✗ *FocusListener/FocusAdapter* – listens for focus gained and focus lost events
- ✗ *KeyListener/KeyAdapter* – listens for key releases and key presses
- ✗ *ModifyListener*(only has 1 method) – listens for text modifications
- ✗ *VerifyListener* – listens for (and potentially intercepts) text modifications
- ✗ ***MouseListener/MouseAdapter*** – listens for mouse button presses
- ✗ *SelectionListener/SelectionAdapter* – listens for selection events (similar to *ActionListener* in Swing)

Simple Example

```
Button loginButton = new Button(shell, SWT.PUSH | SWT.FLAT);  
loginButton.addSelectionListener(new SelectionAdapter() {  
    public void widgetSelected(SelectionEvent e) {  
        System.out.println("Clicked!");  
    }  
});
```

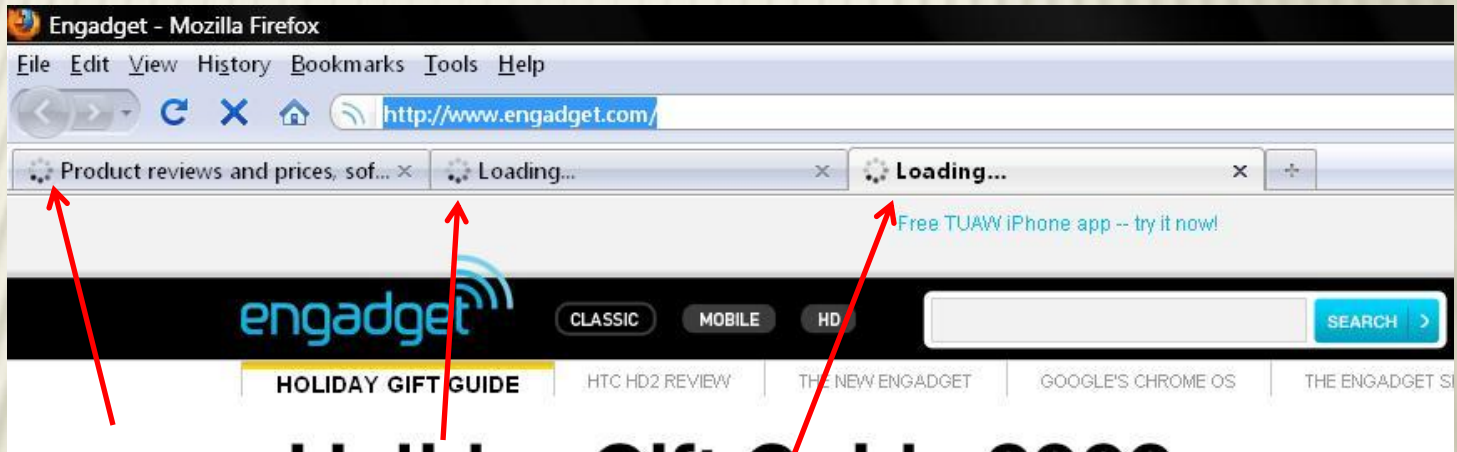
Agenda

- × Project Details
- × SWT
- × Updating UI (Why do we need Threads?)

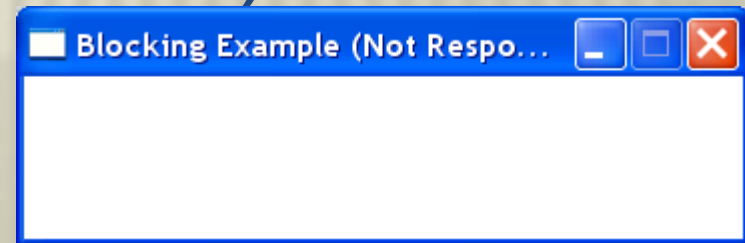
Y do we need threads??

We need threads to support:

- ✖ Multitask applications



- ✖ Long task applications (with GUI)



Threads

- × Context Switch / Interrupt
- × Threads vs Cores (CPUs)
- × Threads vs Processes
(Memory space, “Locking”, Lightweight)

Implementing Threads in Java

2 Options:

- × Implement the Runnable interface: **myImpl**
→ “create a ‘thread class’ and pass **myImpl**”
- × Extend the Thread class: **myClass**
→ “create your **myClass** and start() it

Locks??

- × Why do we need them??
- × No “Mutex”..
- × Define a function as **synchronized**
 - only one thread at a time can “enter it”

More on Java & Threads

- × Really not complicated..

Read link!!

- × <http://www.javabeginner.com/learn-java/java-threads-tutorial>

- × “ThreadPooling PDF” on the course site

Lets start from the end..

- ✖ Update the UI from the UI thread
- ✖ For any other thread, use:
 - `syncExec(Runnable)`
 - `asyncExec(Runnable)`

Going Back, Example for updating UI

```
final Text text = new Text(shell, SWT.BORDER);  
text.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
```

```
Button button = new Button(shell, SWT.PUSH | SWT.FLAT);  
button.setText("Click Me");
```

```
button.addSelectionListener(new SelectionAdapter() {  
    public void widgetSelected(SelectionEvent e) {  
        text.setText(getName());  
    }  
});
```

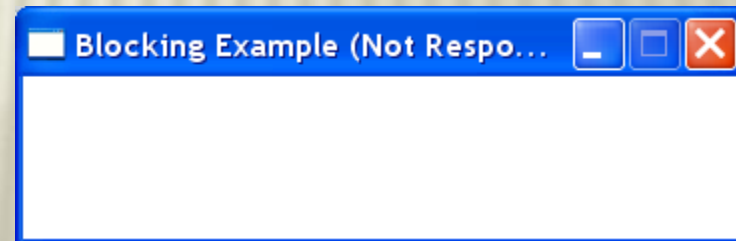
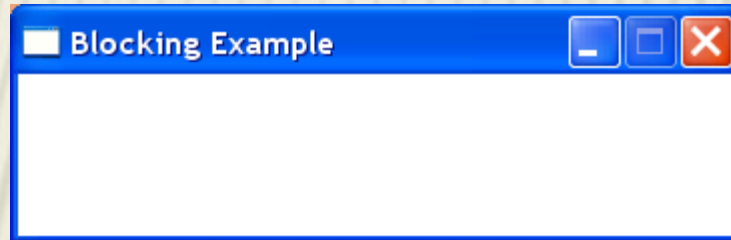
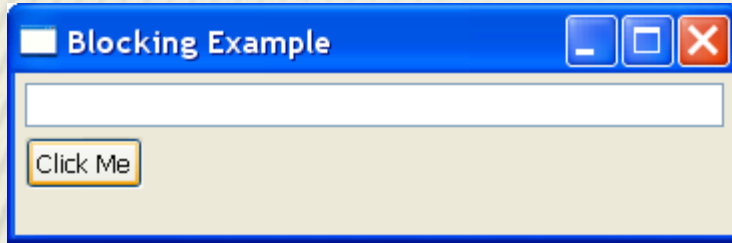
Blocking function

- × “A function that takes long time to return..”
- × Example:
 - While ($i < 1000000000$) {...}
 - Massive DB functions
 - “Slow bandwidth..”

Blocking Function + UI

- ✗ Drawing a UI never ends..
- ✗ The triggered events (e.g. button click) are executed by the drawing thread
- ✗ If the thread is **blocked** for a while, then it can't “draw” the UI and the program “stucks”

Blocking Function + UI



Solution – Threads

- × Use a different thread to calculate getName()

```
public void widgetSelected(SelectionEvent e) {  
    //text.setText(getName());  
    “create thread to calculate getName”  
}
```

- × But who will call “text.setText(“answer”)”?

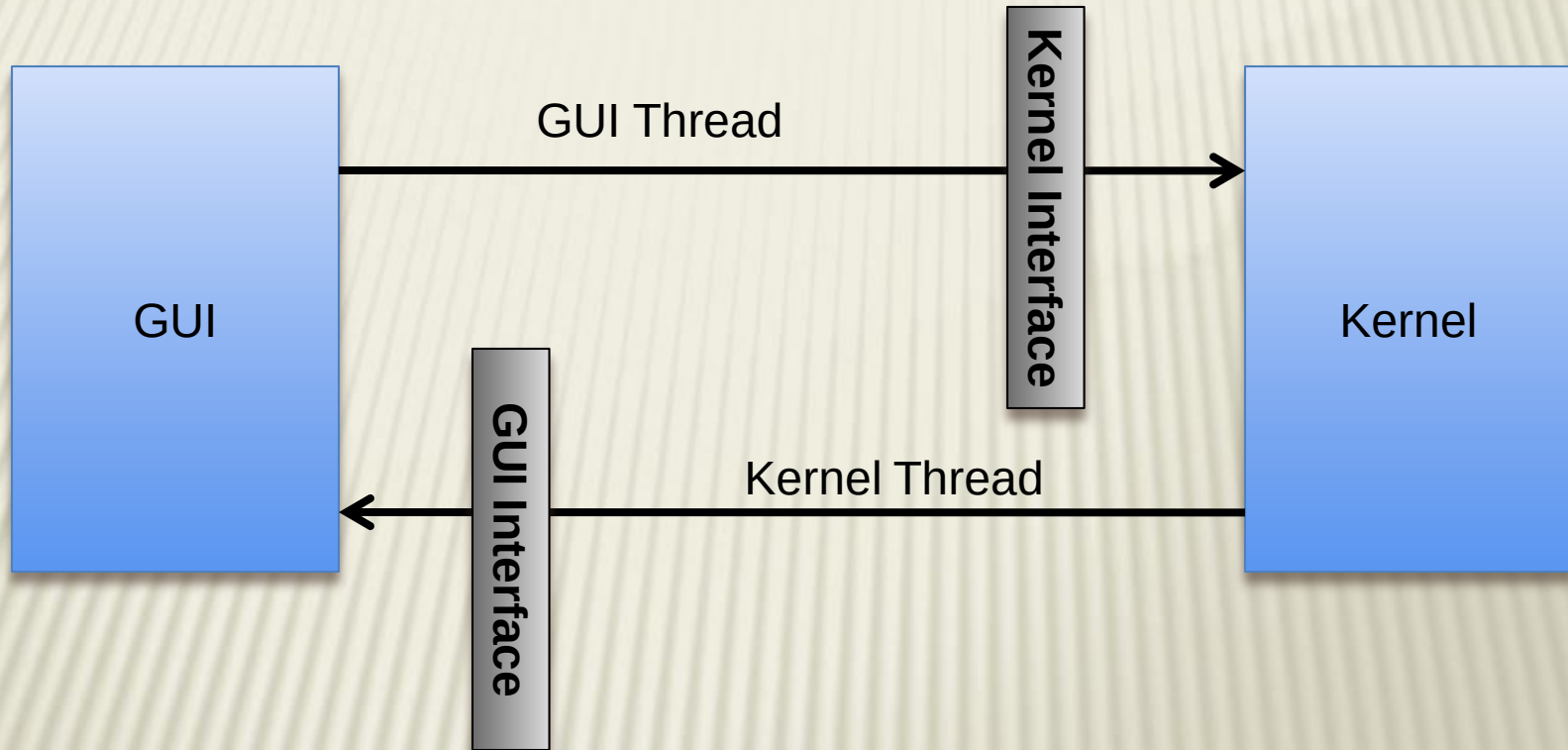
Setting UI from different Thread

- × You **CANNOT** call “text.setText(“answer”)” from the new thread (Exception..)
- × Use:
 - syncExec(Runnable)
 - asyncExec(Runnable)
- × The “sync” blocks until the UI thread updates the UI

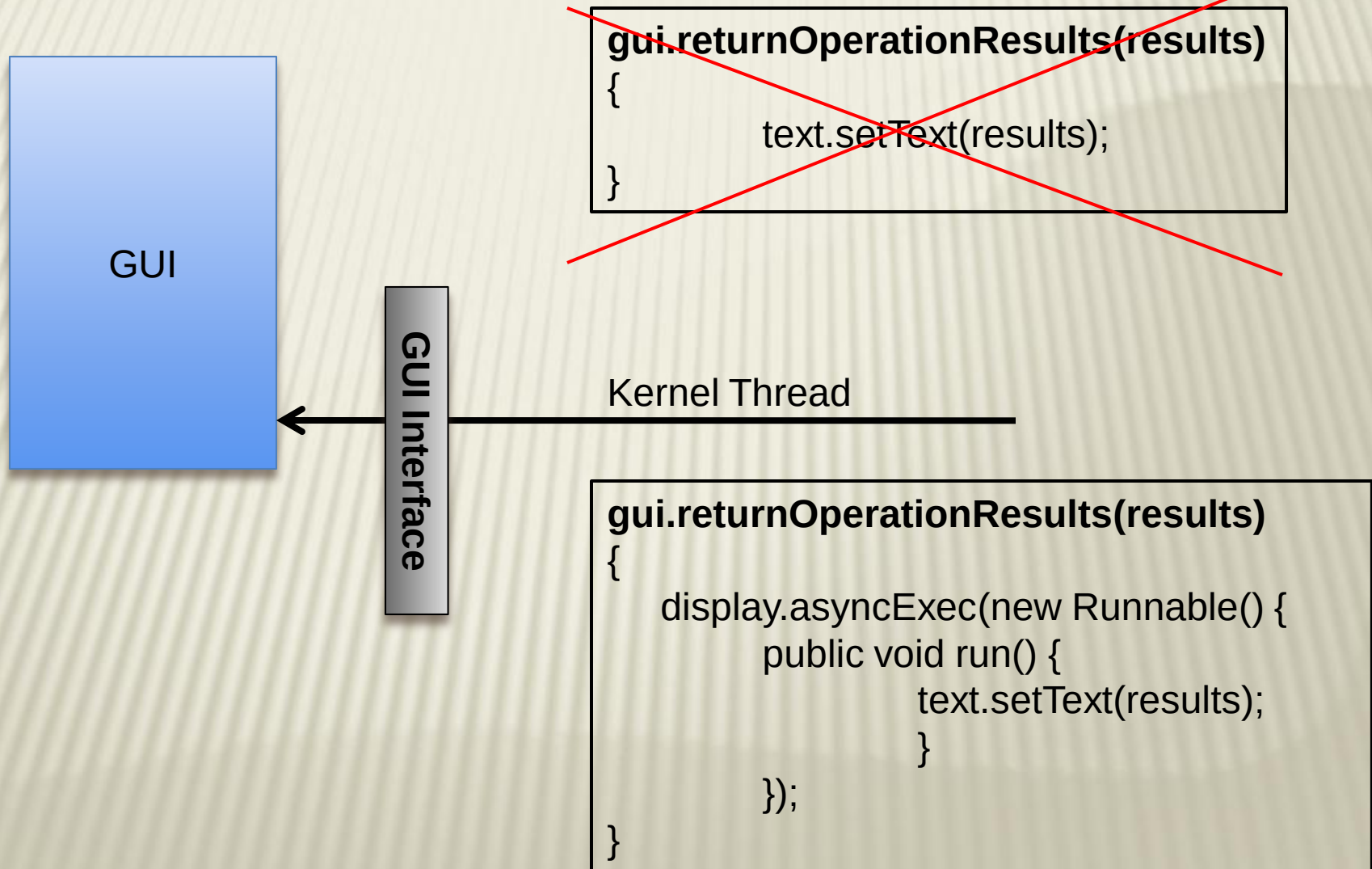
Setting UI from different Thread

```
display.asyncExec(new Runnable() {  
    public void run() {  
        text.setText("answer");  
    }  
});
```

NoneBlocking Layering



NoneBlocking Layering – Updating GUI



Example of NonBlocking Functions (1)

(there are many ways, this is just one)

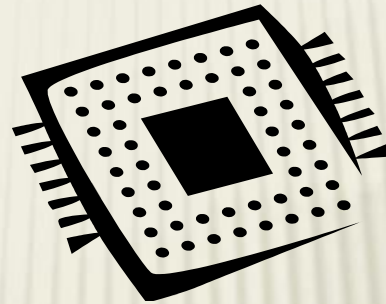
- × Use a “kernel” – i.e. thread manager
- × Implement a pool of Threads
- × Implement a queue of “requests”
- × The results of the function will be return by the pooled thread

Example of NonBlocking Functions (1)



getLongOperation1()

The function returns
immediately with a request id



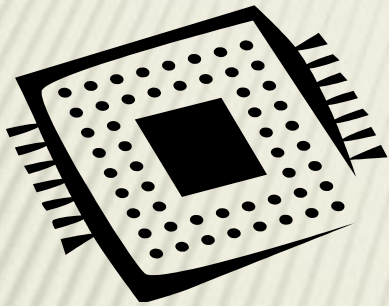
```
int getLongOperation1()
```

```
    requestID      =      "Generate unique ID"  
    addRequestToQueue("LongOperation1", requestID)  
    return requestID
```

```
addRequestToQueue(operation, requestID)
```

```
    queue.add(opration)  
    wakeUpThreads
```

Example of NonBlocking Functions (1)



`returnLongOperation1Results()`



AfterThreadIsAwake()

“Get top request from the queue”

“Process the request”

“return the results to the gui” →:

`gui.returnOperationResults(operation, requestId)`

Example of NonBlocking Functions (2)

- ✖ You can use Java's thread pool / executor
<http://docs.oracle.com/javase/tutorial/essential/concurrency/executors.html>

(The Fork/Join is NOT for you..)

Use a combination of example (1) and (2)

Interesting Questions

- × How to return back the answer
(How to notify back the “right” GUI)
- × How to represents tasks (Consts / Classes)