

DB Programming - Cont

Database Systems

Agenda

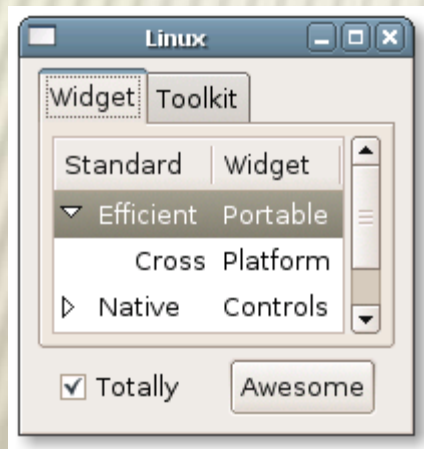
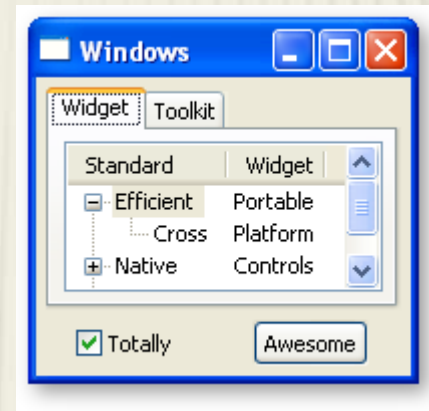
- × SWT

- × Updating the UI (Why do we need Threads?)

SWT (Standard Widget Toolkit)

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

SWT - Takes the look of the OS



Using the SWT package

- ✖ Same as always:

- Add the right swt.jar (Windows/Linux/OS X)

- (<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 Children

Hello World

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

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, it must have a parent:
 - Display (connection to the OS)
- × If you do not keep the shell open (listening to events) the program immediately terminates
- × We can extract the event loop to a utility class

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 Widgets

- ✖ Widgets are created by:
 - Specifying parent
 - Specifying style
- ✖ A parent is the container inside which the widget is created (e.g. Shell)
- ✖ Styles can be specified by constants 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

([javadocs](#))

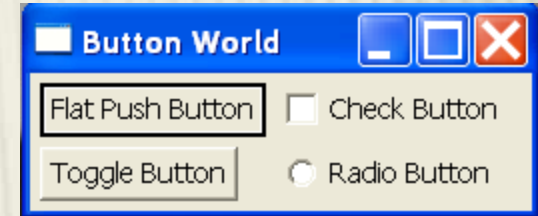
```
Shell shell = SWTUtil.getShell();  
shell.setText("Label World");  
shell.setLayout(new GridLayout()); // layouts are explained later
```



```
// 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

([javadocs](#))

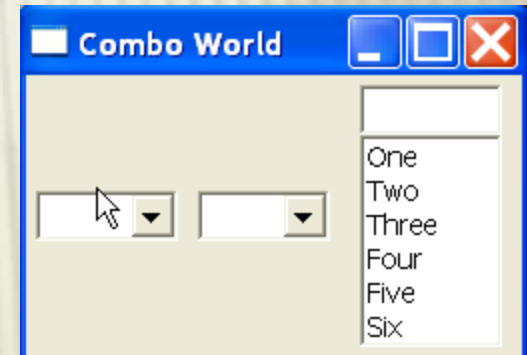
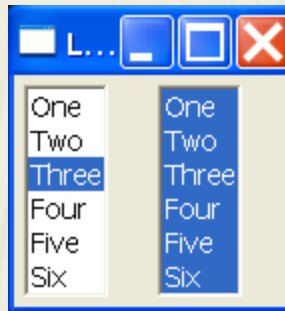


```
shell.setText("Button World");  
shell.setLayout(new GridLayout(2, true)); // layouts are  
explained later
```

```
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)



An example

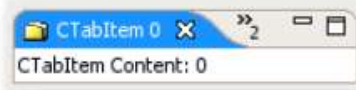
Some more Widgets (2)

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



CoolBar

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



CTabFolder

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



DateTime

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



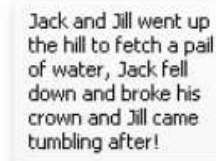
ExpandBar

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



Group

[javadoc](#)



Label

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

Some more Widgets (3)

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



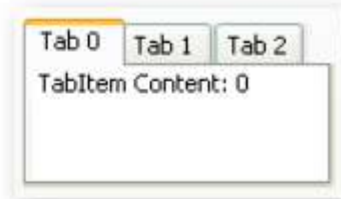
Spinner

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



StyledText

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



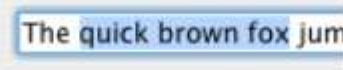
TabFolder

[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

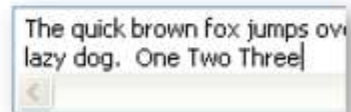
Table

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



Text (SWT.SINGLE)

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



Text (SWT.MULTI)

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

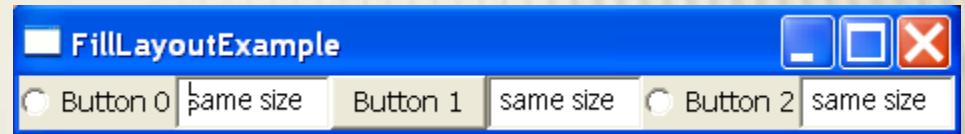
Layouts

- ✖ 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



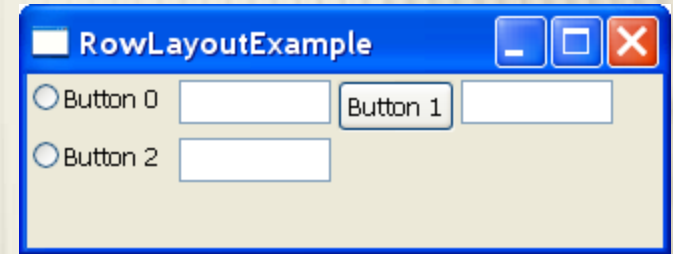
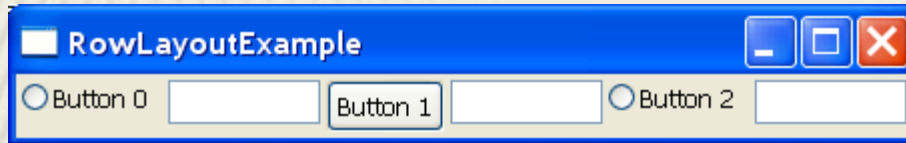
```
shell.setLayout(new FillLayout(SWT.HORIZONTAL));
```

```
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

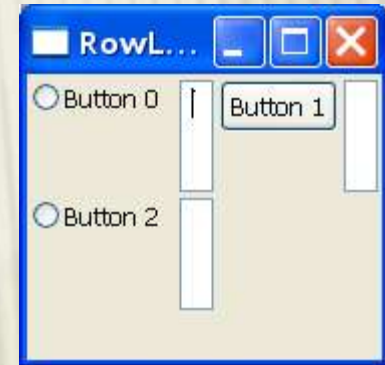
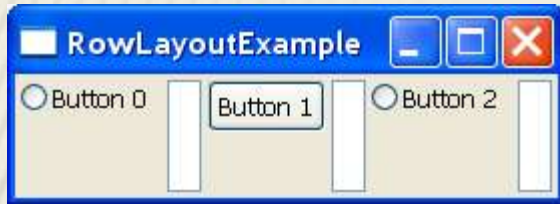
RowLayout



```
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

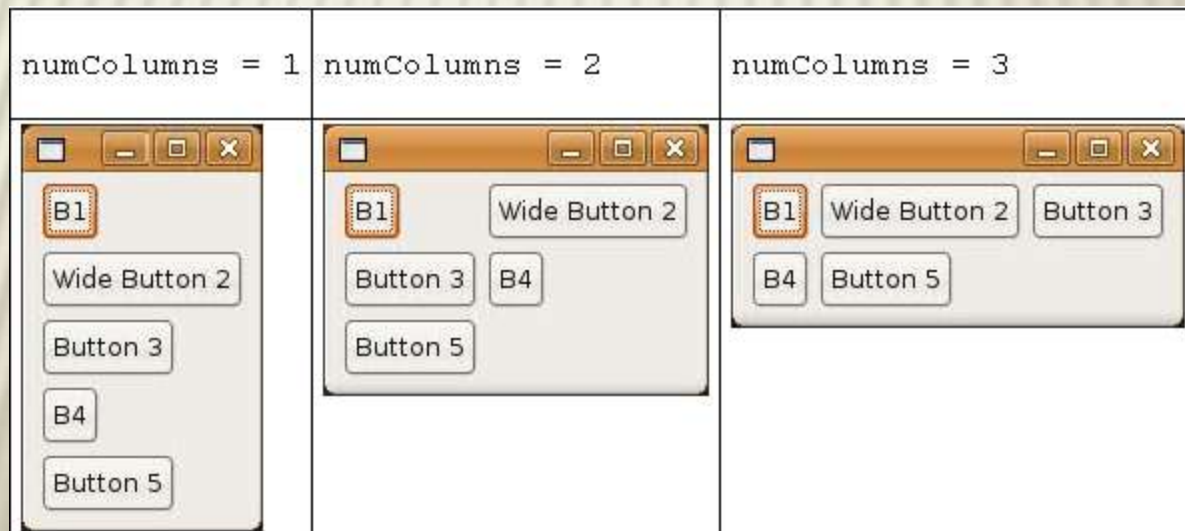


```
shell.setLayout(new RowLayout(SWT.HORIZONTAL));
```

```
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

- × Provides 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 2x2 grid of buttons. The buttons are labeled B1, Wide Button 2, Button 3, and B4. Button 5 is also present. The buttons are aligned to the left (BEGINNING) within their grid cells.
<pre>horizontalAlignment = GridData.CENTER</pre>	 A GUI window with a 2x2 grid of buttons. The buttons are labeled B1, Wide Button 2, Button 3, and B4. Button 5 is also present. The buttons are centered horizontally within their grid cells.
<pre>horizontalAlignment = GridData.END</pre>	 A GUI window with a 2x2 grid of buttons. The buttons are labeled B1, Wide Button 2, Button 3, and B4. Button 5 is also present. The buttons are aligned to the right (END) within their grid cells.
<pre>horizontalAlignment = GridData.FILL</pre>	 A GUI window with a 2x2 grid of buttons. The buttons are labeled B1, Wide Button 2, Button 3, and B4. Button 5 is also present. The buttons are stretched to fill their grid cells horizontally.

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);
```



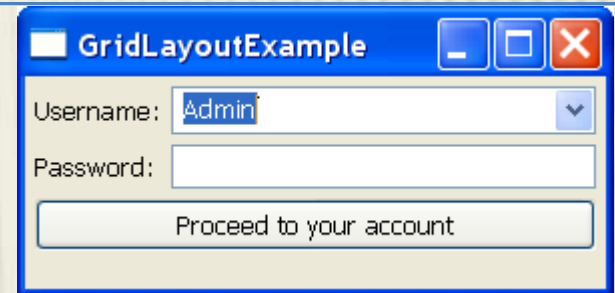
Another Example

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

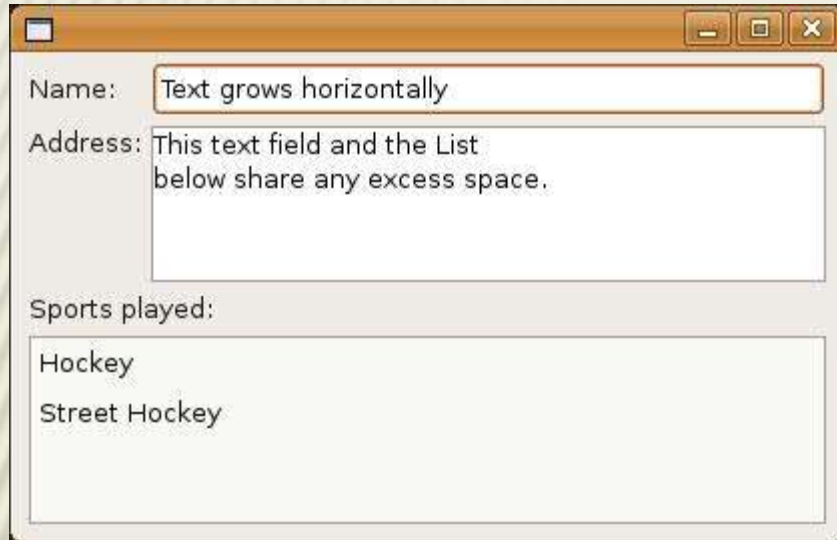
```
new Label(shell, SWT.NONE).setText("Username:");  
Combo cmbUsername = new Combo(shell, SWT.DROP_DOWN);  
cmbUsername.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));  
cmbUsername.setItems(new String[]{"Howard", "Admin", "Kalman"});  
cmbUsername.setText("Admin");
```

```
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);
```



Google for other examples



A screenshot of a simple graphical user interface window. It contains three main sections: a 'Name:' label followed by a single-line text field containing 'Text grows horizontally'; an 'Address:' label followed by a multi-line text area containing 'This text field and the List below share any excess space.'; and a 'Sports played:' label followed by a list box containing '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 includes fields for 'Dog's Name' (Bifford), 'Breed' (Black Lab), and a 'Photo' section with a 'Browse...' button and a 'Delete' button. A 'Categories' list box contains options like 'Best of Breed', 'Prettiest Female', and 'Most Hair'. An 'Owner Info' section has fields for 'Name' (Mary Smith) and 'Phone' (123-4567). An 'Enter' button is 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>

Event Handling

- × **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 presses and releases
- ✗ *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, for example button presses (similar to *ActionListener* in Swing)

Simple Example

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

Agenda

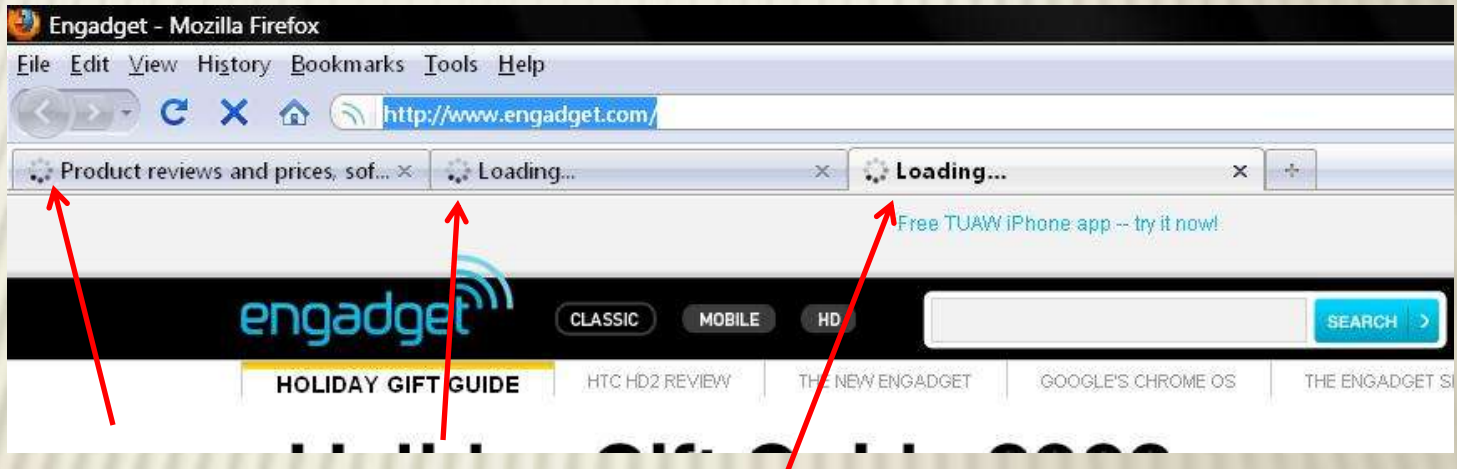
- × SWT

- × Updating the UI (Why do we need Threads?)

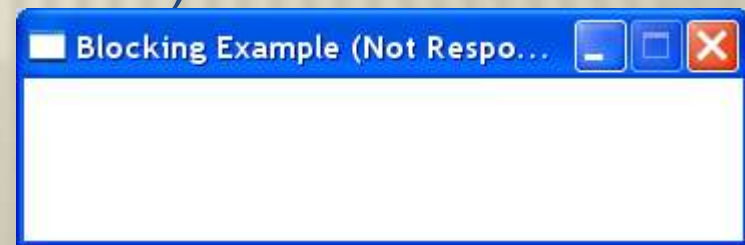
Why do we need threads??

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”
 - per object or per class

More on Java & Threads

Read the links:

- × <http://www.javabeginner.com/learn-java/java-threads-tutorial>
- × <http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>

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());
    }
});
```

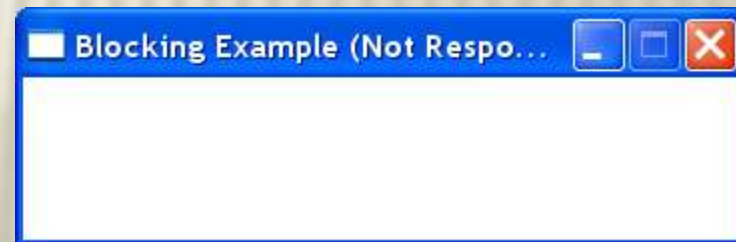
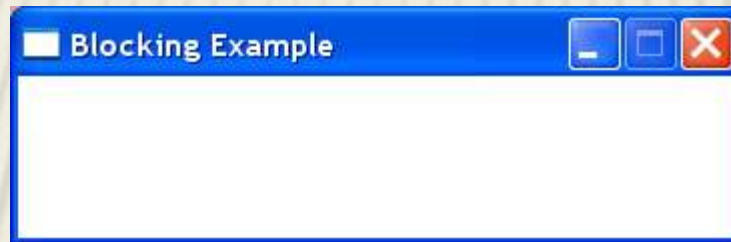
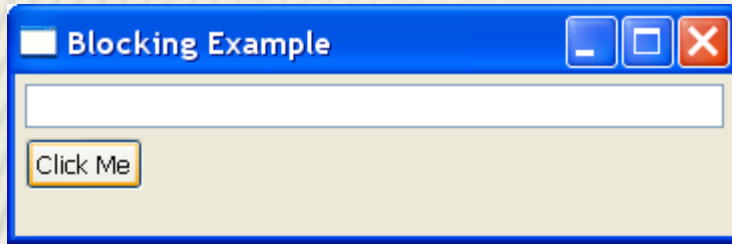
Blocking function

- × “A function that takes a long time to return..”
- × Example:
 - While ($i < 1000000000$){...}
 - DB calls
 - Network communications

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 is “stuck”

Blocking Function + UI



Solution – Threads

- ✖ Use a different thread to calculate getName()

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

- ✖ But who will call text.setText(...)?

Setting the UI from a different Thread

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

Setting the UI from a different Thread

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

Example of Non-Blocking Functions (1)

(there are many ways, this is just one)

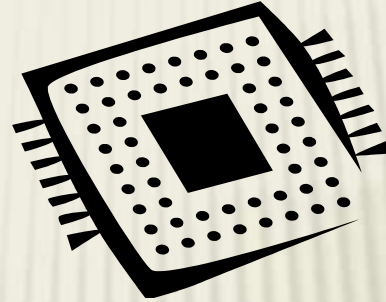
- × Implement a pool of Threads
- × Implement a queue of “requests”
- × The results of the function will be returned by the pooled thread

Example of Non-Blocking Functions (2)



getLongOperation1()

The function returns
immediately with a request id



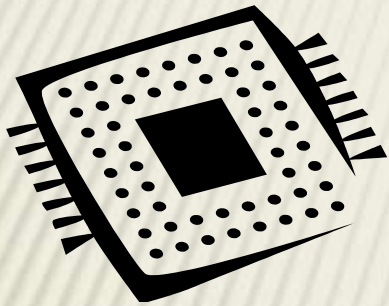
```
int getLongOperation1()
```

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

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

```
    queue.add(operation)  
    wakeUpThreads
```

Example of Non-Blocking Functions (3)



`returnLongOperation1Results()`



`AfterThreadIsAwake()`

“Get top request from the queue”

“Process the request”

“return the results to the GUI” →:

`gui.returnOperationResults(operation, requestID)`

Example of Non-Blocking Functions (4)

- ✖ 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..)

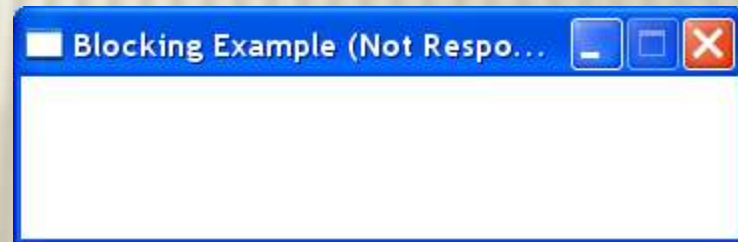
- ✖ Many implementation details are left for you to resolve
 - + How to return the answer
(How to notify the GUI)
 - + How to represents tasks (Constants / Objects)

Bad examples

- × The following next slides are examples for what **NOT-TO-DO** when you write an application (your project, in particular!).
- × Based on last years submissions.

Bad examples: Usability

- × “non-refreshed” windows
- × “Hanging” windows
- × Instead: “Please wait, your query may take a couple of minutes...”, progress bar, etc.



Bad examples: Usability

- ✗ Window layout should be clear and intuitive

The diagram illustrates a search application window with several usability issues. The layout is cluttered and lacks a clear visual hierarchy. The 'Disc search' section is a large container with a 'Title' input field, a 'Year' input field, a 'Genre' input field, and an 'On sale' checkbox. A 'New search' button is positioned at the top right of the window. Below the 'Disc search' section is a 'Track search' section with a 'Title' input field. At the bottom of the window are three buttons: 'connect', 'close', and 'search'. The 'search' button is underlined with a green wavy line. Below these buttons is a 'Result table:' label followed by a large empty rectangular box for the results.

New search

Disc search

Title

Year

Genre

On sale

Track search

Title

connect

close

search

Result table:

Bad examples: Usability

- ✗ Window layout should not be overloaded
- + Use tabs, pop-up dialogs, drop down menus...

<http://www.ssw.com.au/ssw/Standards/Rules/RulesToBetterinterfaces.aspx>

