DB Programming - Cont

Database Systems

Agenda

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

Agenda

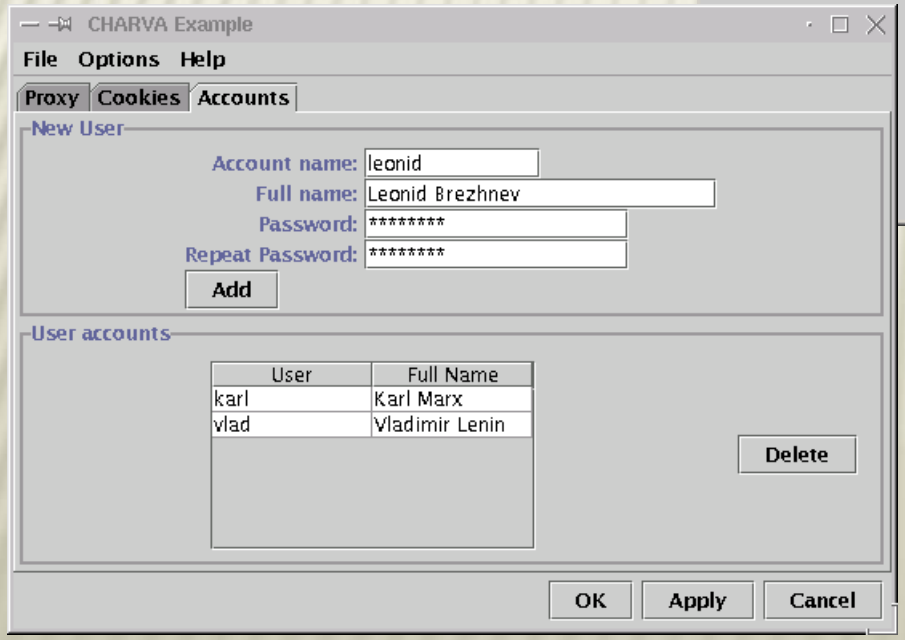
- Project Details
- SWT
- Updating UI (Why do we need Threads?)

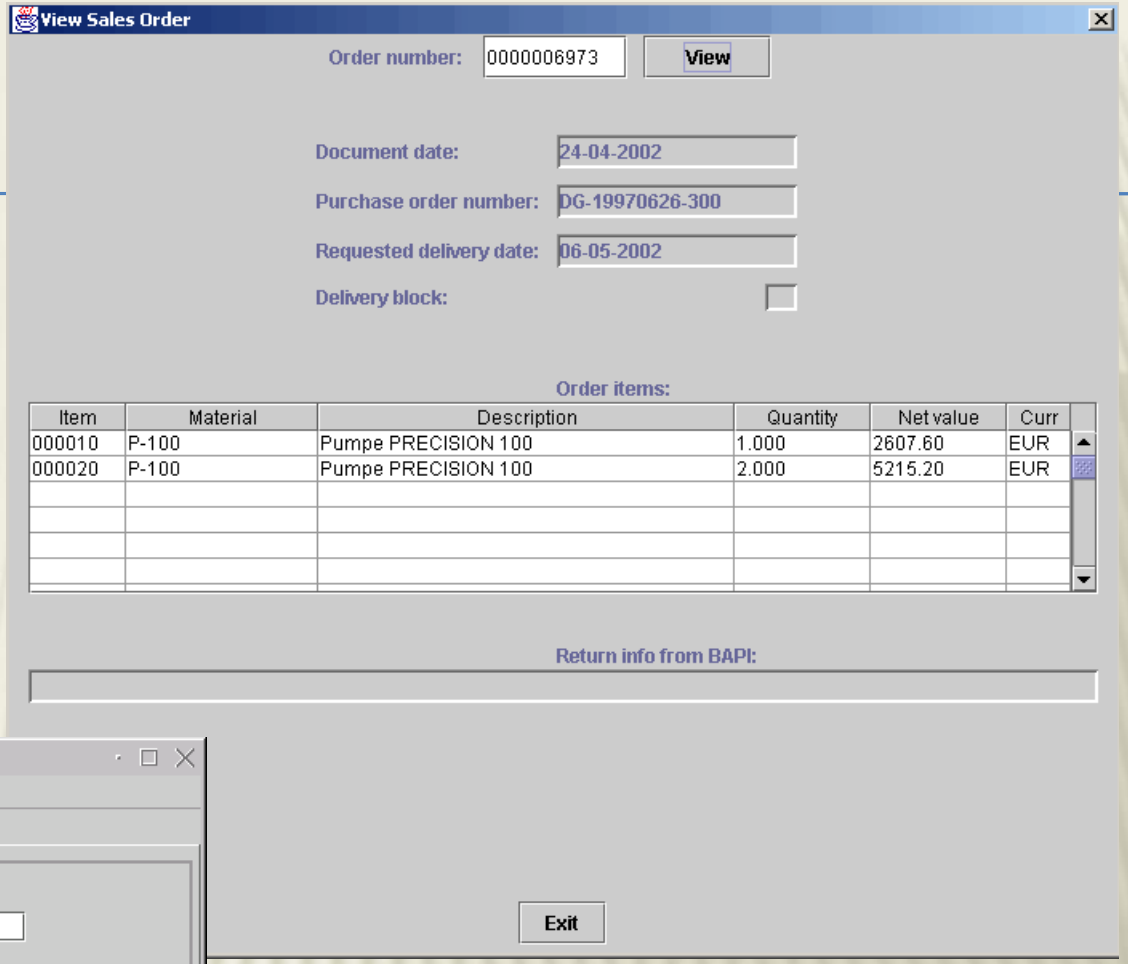
Other JAVA UI

- AWT (Abstract Windowing Toolkit)
  - standard for all platforms
  →too simple..
  - “Least Common Denominator”

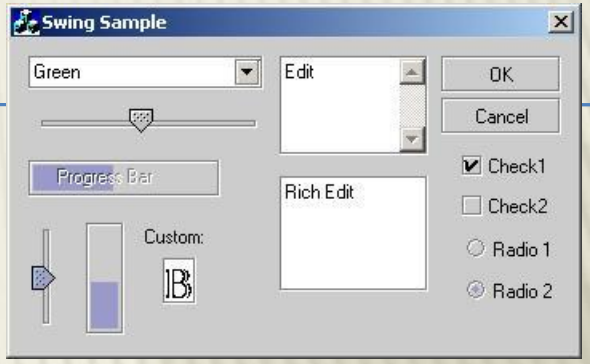
- SWING
  - try to fix AWT’s problems
  - uses AWT
  - complicated to use and learn
  - looks the same on every platform

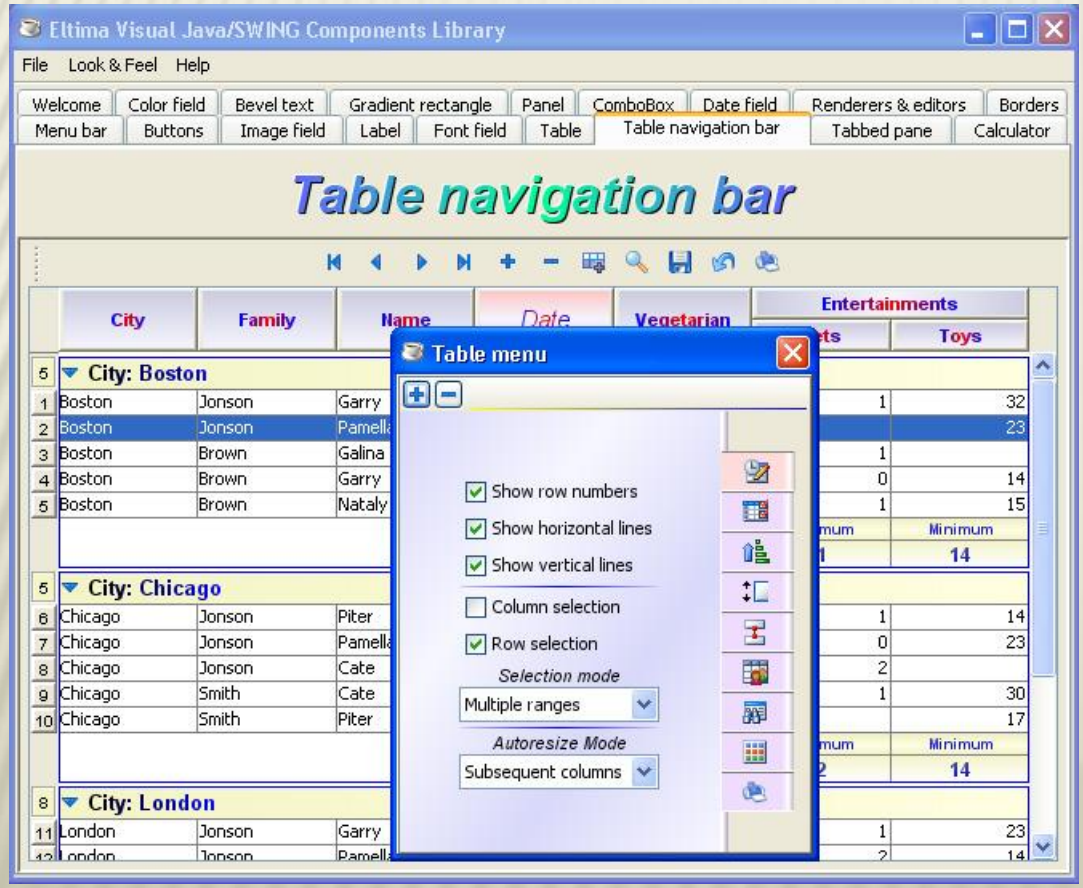
AWT


CHARVA Example
File Options Help
Proxy Cookies Accounts
New User
Account name: leonid
Full name: Leonid Brezhnev
Password: ********
Repeat Password: ********
Add
User accounts
User Full Name
karl Karl Marx
vlad Vladimir Lenin
Delete
OK Apply Cancel


View Sales Order
Order number: 0000006973 View
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:
Exit

Swing

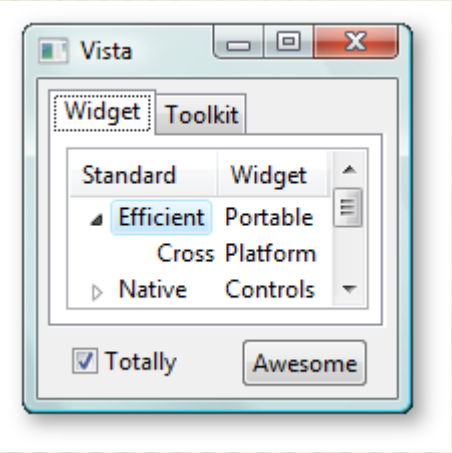

Swing Sample
Green
Edit
OK
Cancel
Progress Bar
Rich Edit
Check1
Check2
Radio 1
Radio 2
Custom:

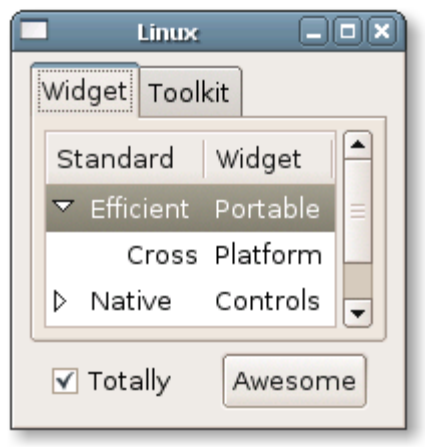

Eltima Visual Java/SWING Components Library
File
Look & Feel
Help
Welcome
Color field
Bevel text
Gradient rectangle
Panel
ComboBox
Date field
Renderers & editors
Borders
Menu bar
Buttons
Image field
Label
Font field
Table
Table navigation bar
Tabbed pane
Calculator
Table navigation bar
City
Family
Name
Date
Vegetarian
Entertainments
Toys
City: Boston
City: Chicago
City: London
Table menu
Show row numbers
Show horizontal lines
Show vertical lines
Column selection
Row selection
Selection mode
Multiple ranges
Autoresize Mode
Subsequent columns
Minimum

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





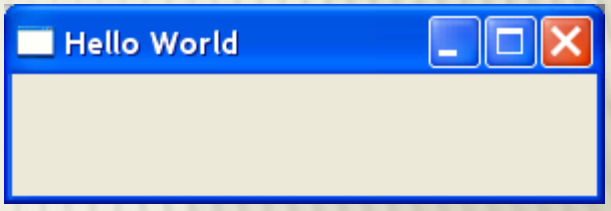
“Installing” SWT

- 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

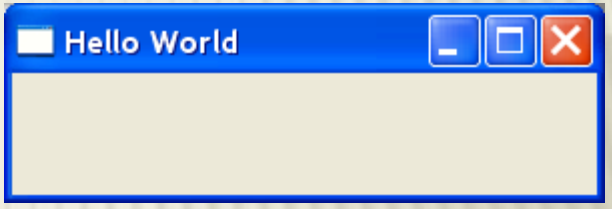
- Can we have a more “swing” environment?

SWTUtill

```
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 SWTUtill)

```
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

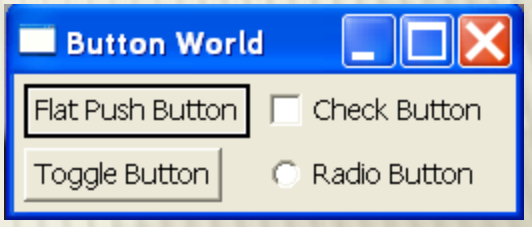
- A parent is the container that the widget is created inside (e.g. Shell)
- A style is a constant from the SWT class (SWT.BORDER, SWT.LEFT, SWT.NONE …)

- 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);
```

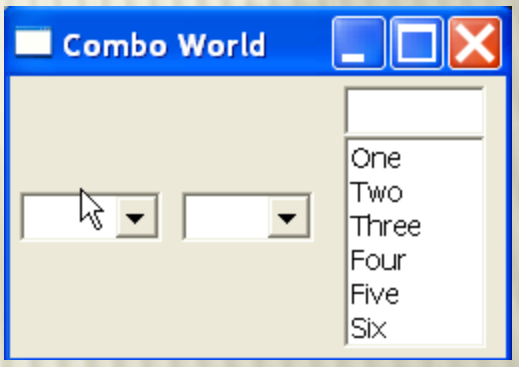
Button

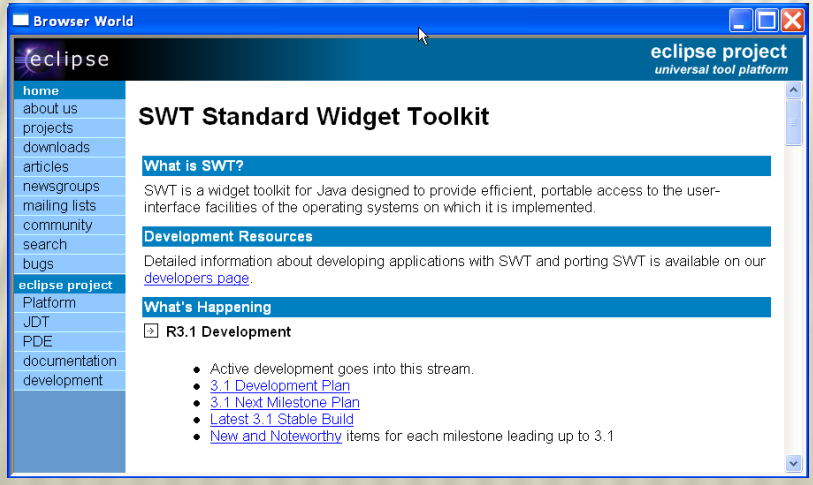



```
shell.setText("Button World");
shell.setLayout(new GridLayout(2, true)); // layouts are explained later

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)



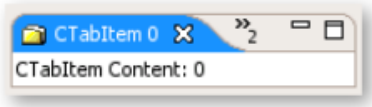


Who's your daddy??

Some more Widgets (2)

- http://www.eclipse.org/swt/widgets/

CoolBar
javadoc - snippets

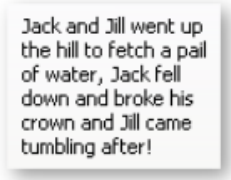


CTabFolder
javadoc - snippets

DateTime
javadoc - snippets

ExpandBar
javadoc - snippets

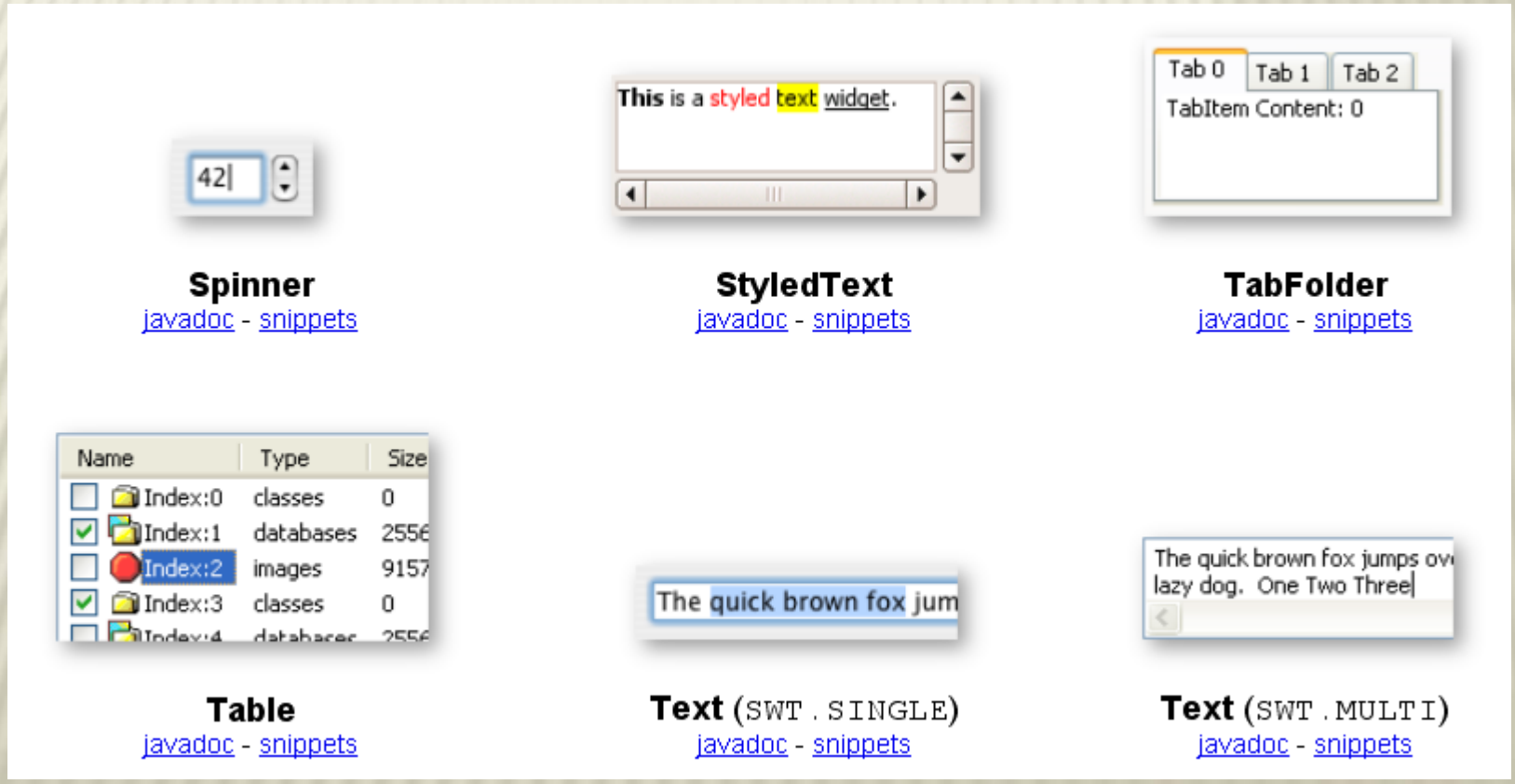
Group
javadoc



Label
javadoc - snippets

Some more Widgets (3)

- http://www.eclipse.org/swt/widgets/



Layouts

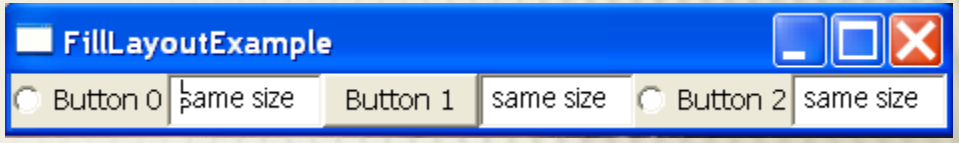
- First introduced in AWT
- Ease burden of laying out components
- SWT offers 5 layouts:
  - FillLayout
  - RowLayout
  - GridLayout
  - FormLayout
  - (*) Stack Layout

- 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

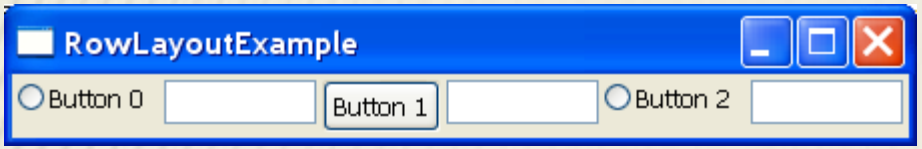


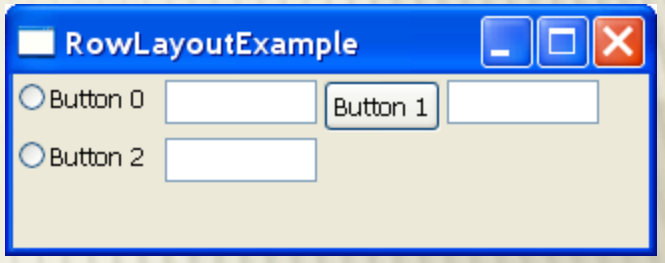

```
shell.setLayout(new FillLayout(SWT.HORIZONTAL));

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

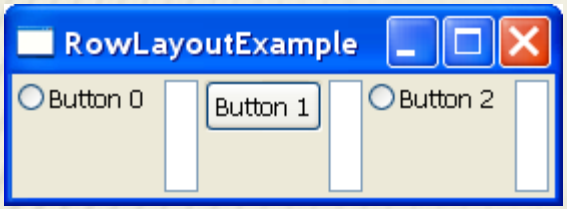
RowLayout

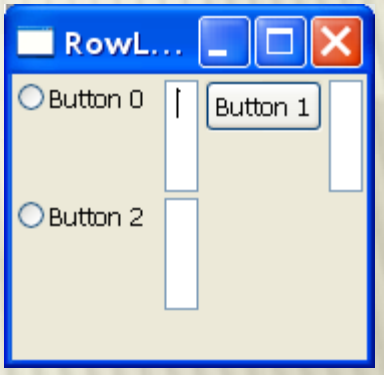





```
shell.setLayout(new RowLayout(SWT.HORIZONTAL));

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

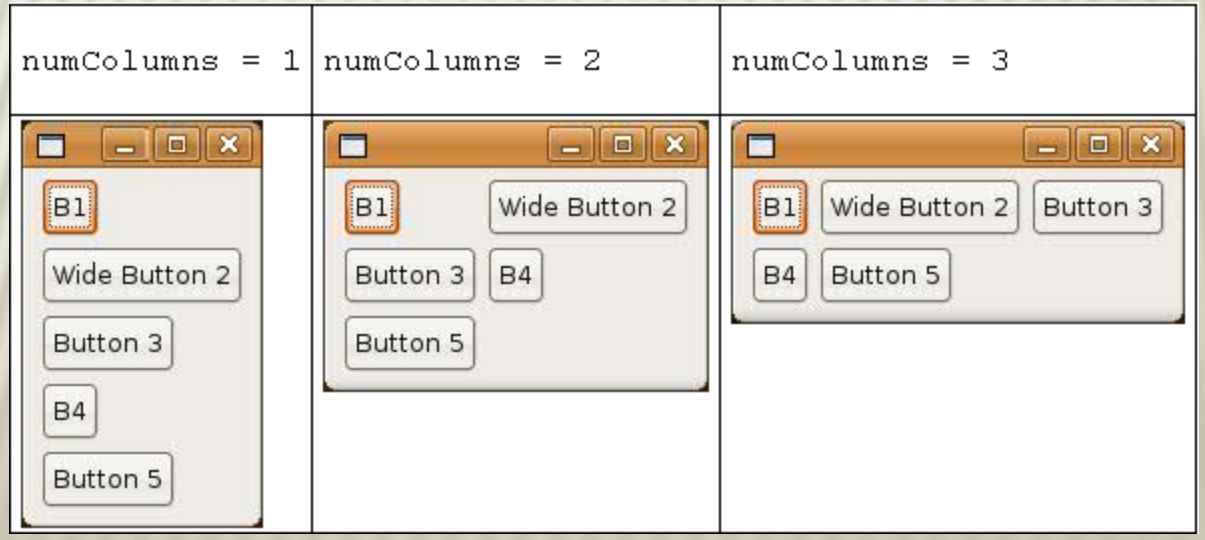





```
shell.setLayout(new RowLayout(SWT.HORIZONTAL));

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

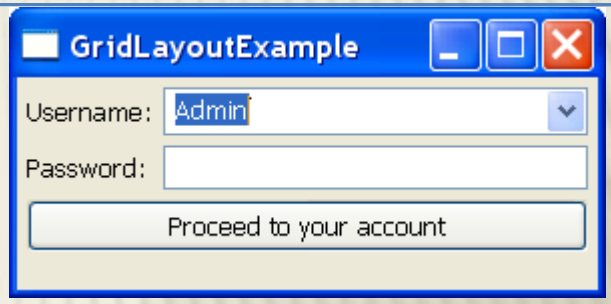
GridData

- 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

`horizontalAlignment = GridData.BEGINNING` (default)	
`horizontalAlignment = GridData.CENTER`	
`horizontalAlignment = GridData.END`	
`horizontalAlignment = GridData.FILL`	

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);
```

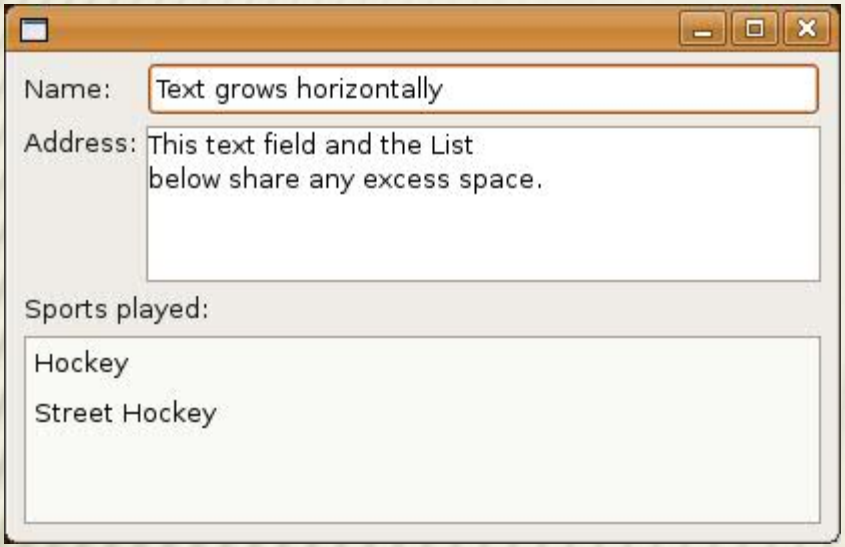
Another Example

```
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



- 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!");
    }
});
```

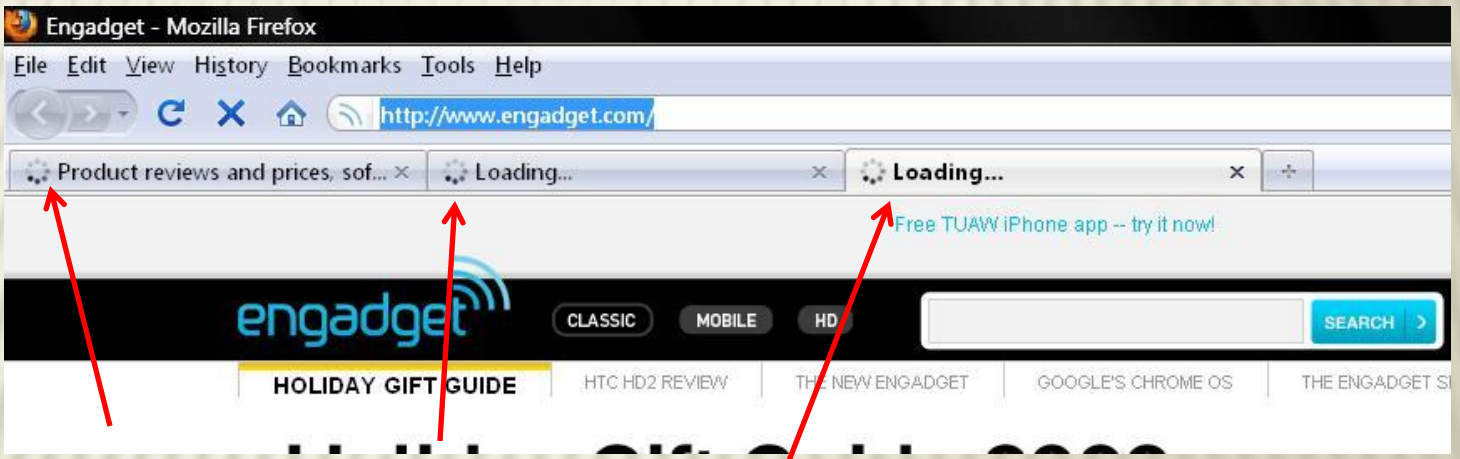
Agenda

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

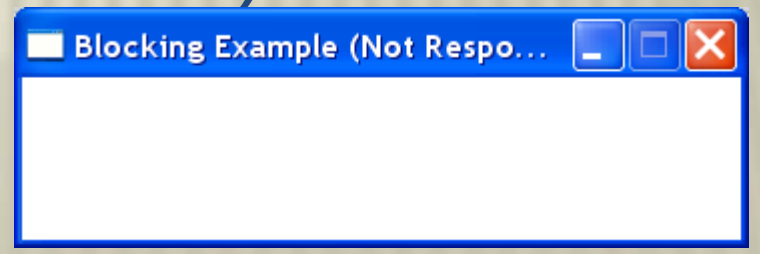
Y do we need threads??

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

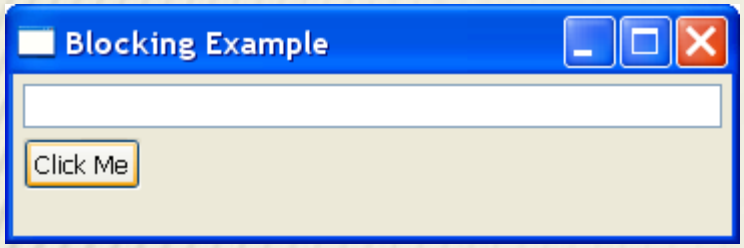
- “A function that takes long time to return..”

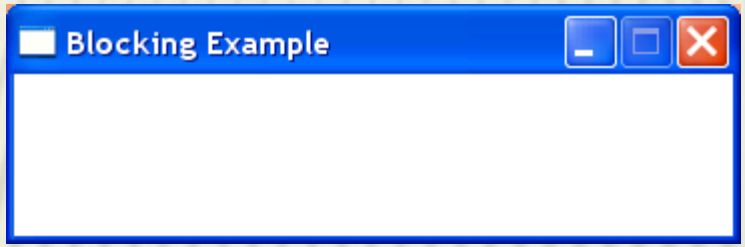
- Example:
  - While (i<100000000){…}
  - Massive DB functions
  - “Slow bandwidth..”

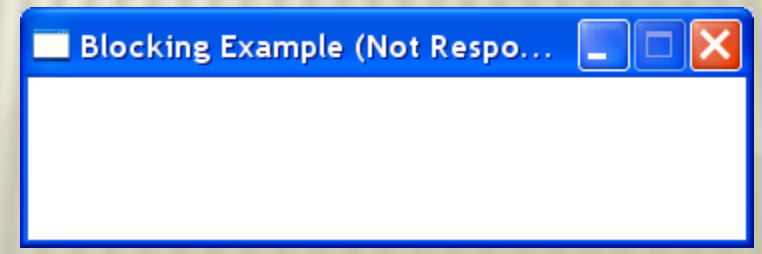
Blocking Function + UI

- 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..)

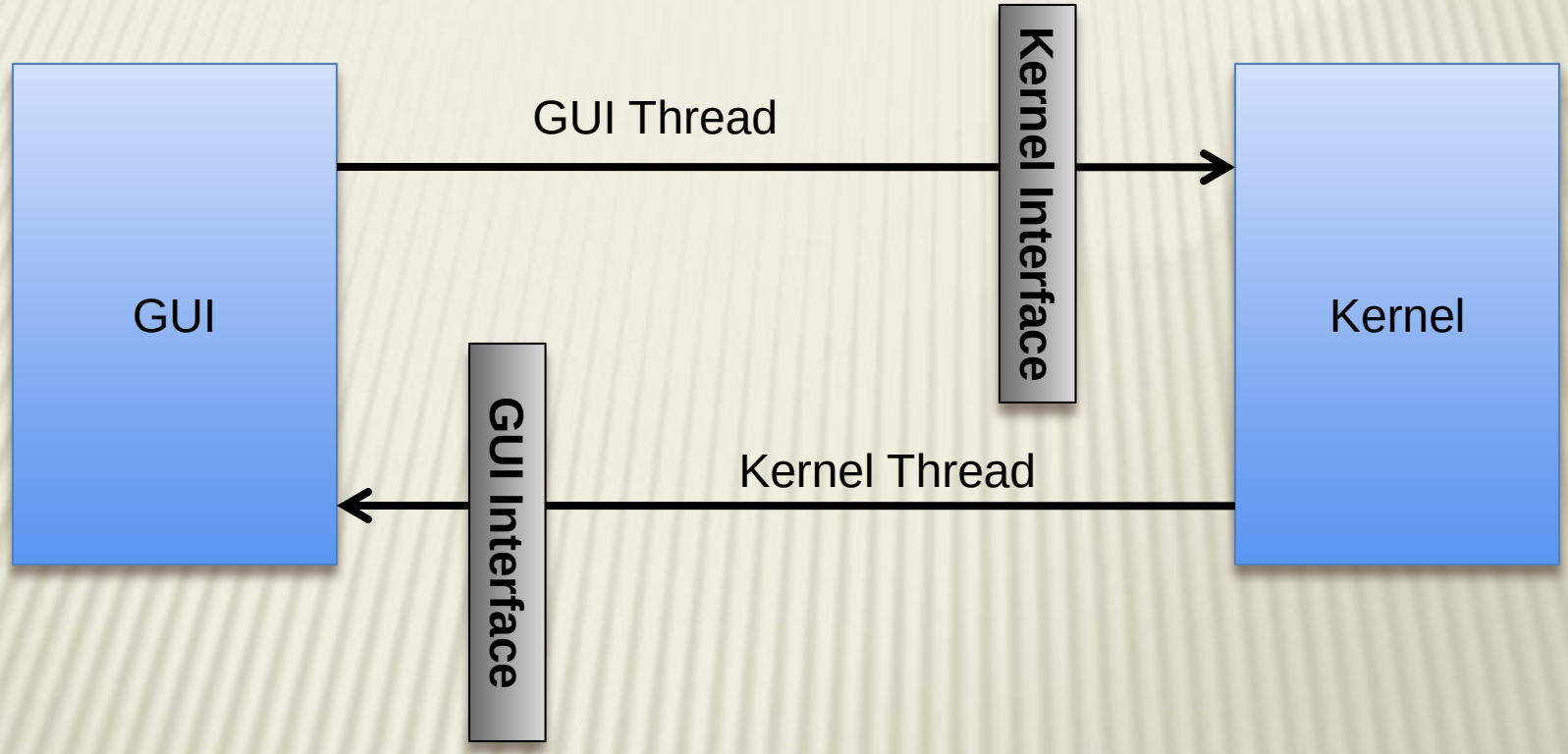
- Use:
  - syncExec(Runnable)
  - asyncExec(Runnable)

- The “sync” blocks until the UI thread updates the UI

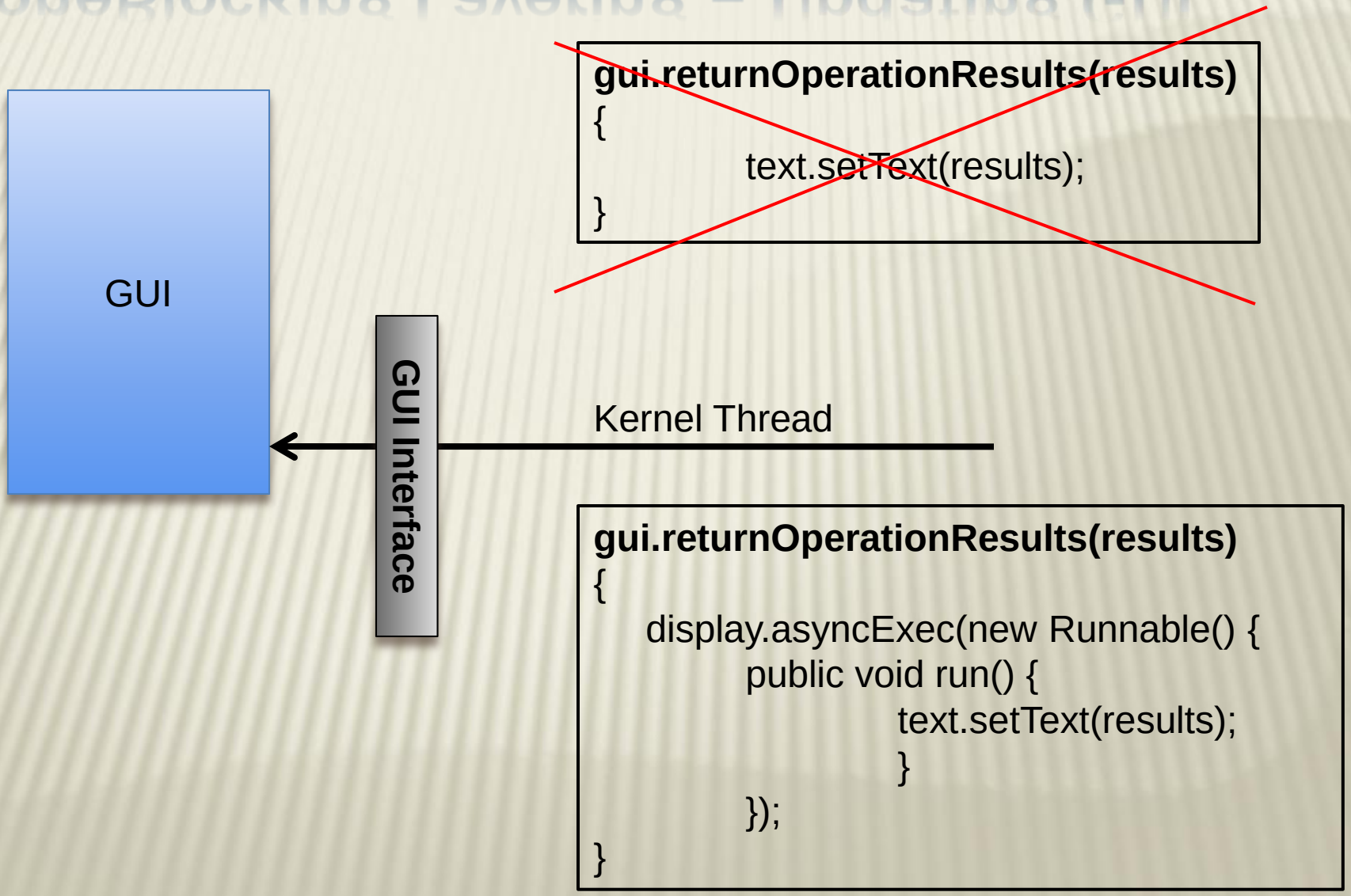
Setting UI from different Thread

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

NoneBlocking Layering



NoneBlocking Layering – Updating GUI



Example of NoneBlocking Functions (1)

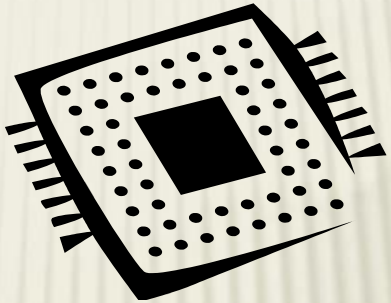
(there are many ways, this is just one)

- 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 NoneBlocking Functions (1)

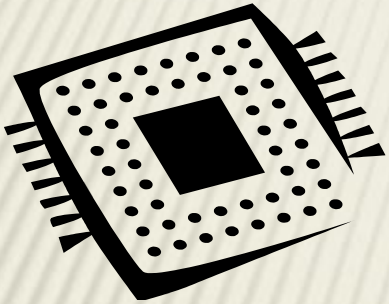
getLongOperation1()

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 NoneBlocking 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 NoneBlocking Functions (2)

- You can use Java's thread pool / executor
  http://java.sun.com/docs/books/tutorial/essential/concurrency/exinter.html
  http://java.sun.com/docs/books/tutorial/essential/concurrency/pools.html

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)