Exception Handling in Multi-Layered Systems

Layers and Exceptions

Christian Rehn

Delphi-Treff

ADUG Sydney Meeting
21st November 2012

Who Am I?

- Christian Rehn
- CS Student at the University of Kaiserslautern
- Moderator and editor for Delphi-Treff (some German Delphi website)
- `http://www.christian-rehn.de/`

[1] In contrast to the rest, the logos, of course, aren't CC licensed.

Organisational Stuff

- A basic understanding of OOP is needed
- Few text on the slides
  - Better for presentation
  - There are more detailed talk notes online:
    `http://www.christian-rehn.de/`
  - German version is even more detailed but I haven't had the time to translate everything
- Please give feedback (what can I do better?)

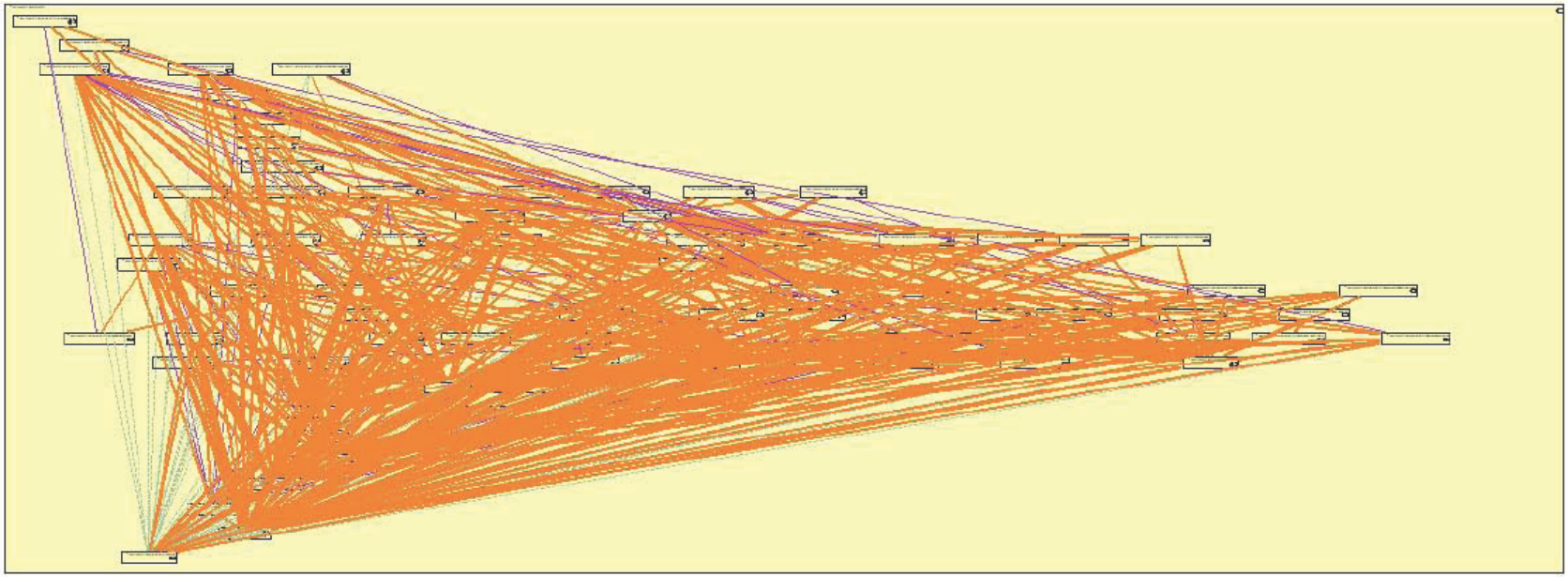
Overview

Motivation

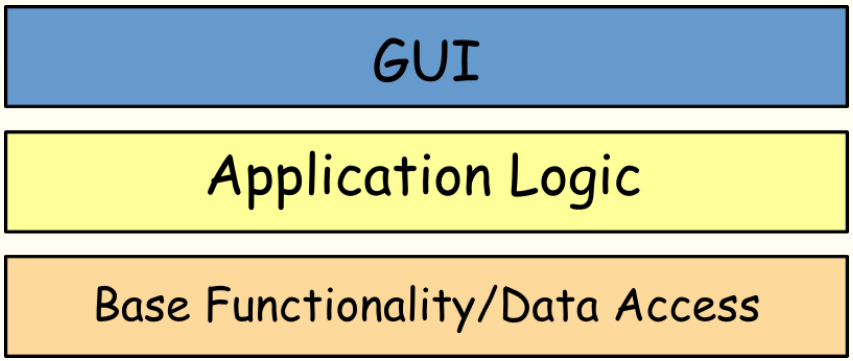
Why Think? (1/2)

Why should we think about all that?

Why Think? (2/2)



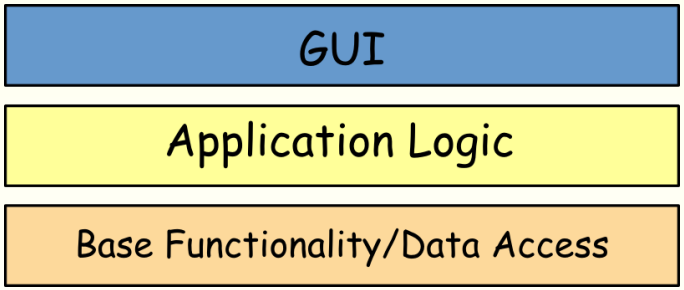
Layers



Why Exceptions?

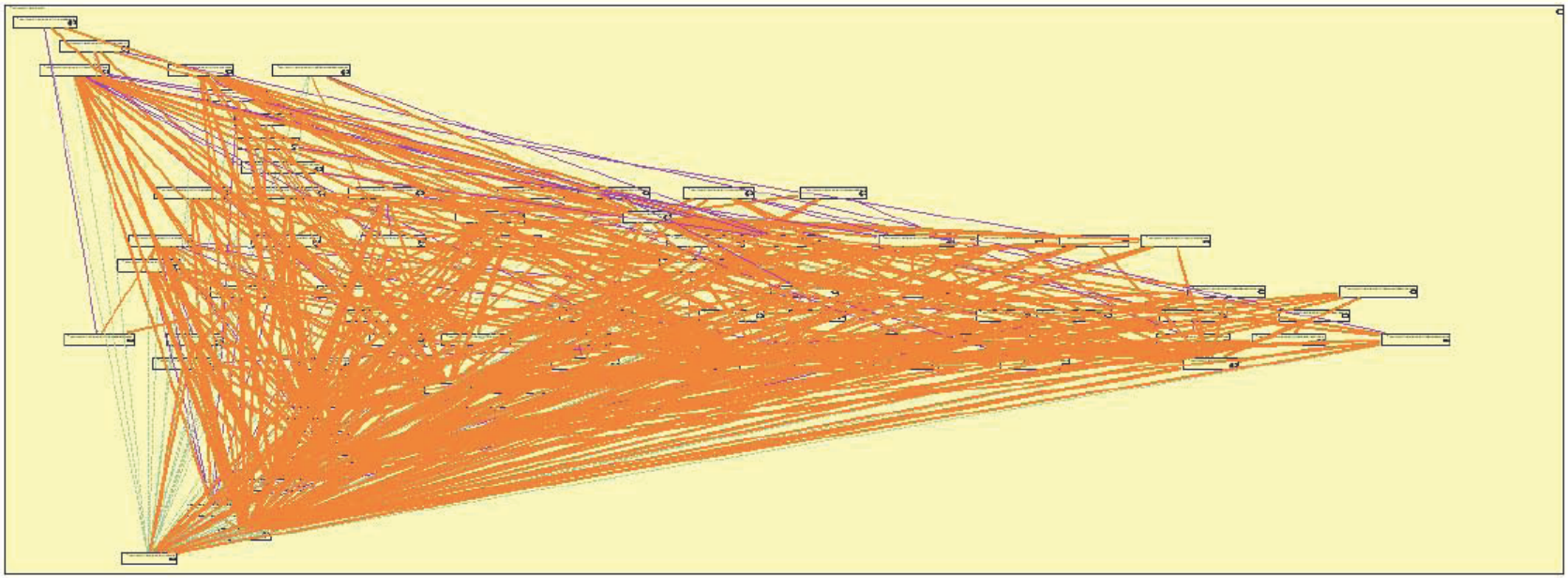
```
try
  while not EndOfTalk do
  begin
    Present( slide );
    GoToNextSlide;
  end;
except
  on e: EFireAlarm do
  begin
    Panic;
    Shout(e.Message);
  end;
end;
```

And what's the link between these two topics?




```
on e: EFireAlarm do
begin
  Panic;
  Shout(e.Message);
end;
```

Dependencies and Layers



Ripple Effects

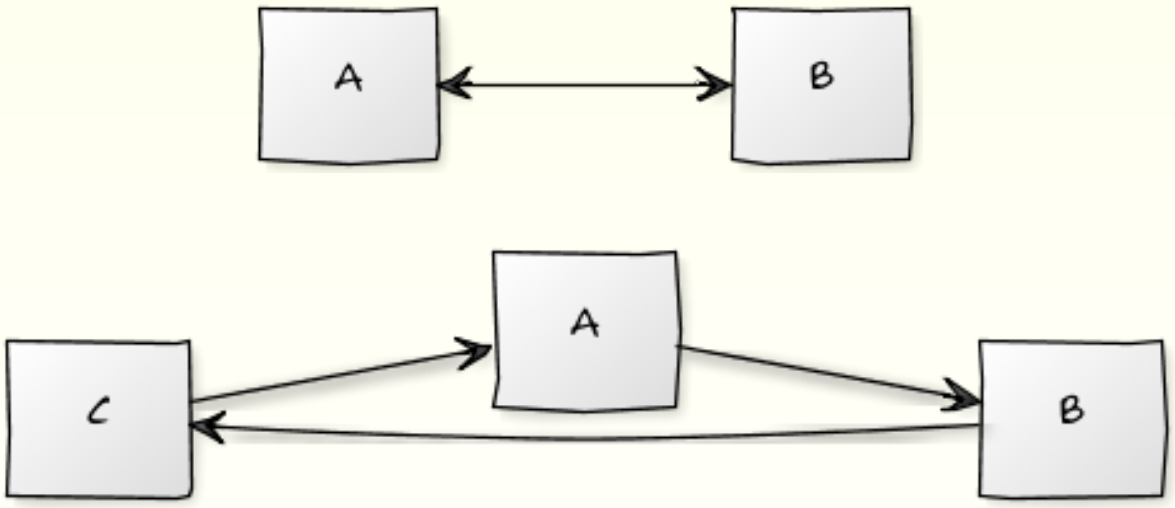
2

[2] CC-BY-SA Rainer Zenz http://commons.wikimedia.org/wiki/File:2006-01-14_Surface_waves-2.jpg

Dependencies

```
myFancyOpenDialog.ShellTreeView.Path := pathToMyDocuments;
myFancyOpenDialog.FileNameEdit.Text := 'newFile.txt';
 if  myFancyOpenDialog.ShowModal = mrOK then
begin
   pathToSaveFile := myFancyOpenDialog.ShellTreeView.Path + myFancyOpenDialog.FileNameEdit.Text;
   SomeMemo.Lines.LoadFromFile(pathToSaveFile);
end;
```

Cyclic Dependencies



Circular Unit References

```
[DCC Fatal Error] UnitXY.pas(7):  F2047 Circular unit
reference to 'UnitXY'
```

Architecture

Architecture: Definition by SEI

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [BCK03]

Architecture: My Definition

The software architecture describes the coarse structures of the software and defined how to *think* about it as a developer.

Architecture

Architecture: Definition by SEI

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [BCK03]

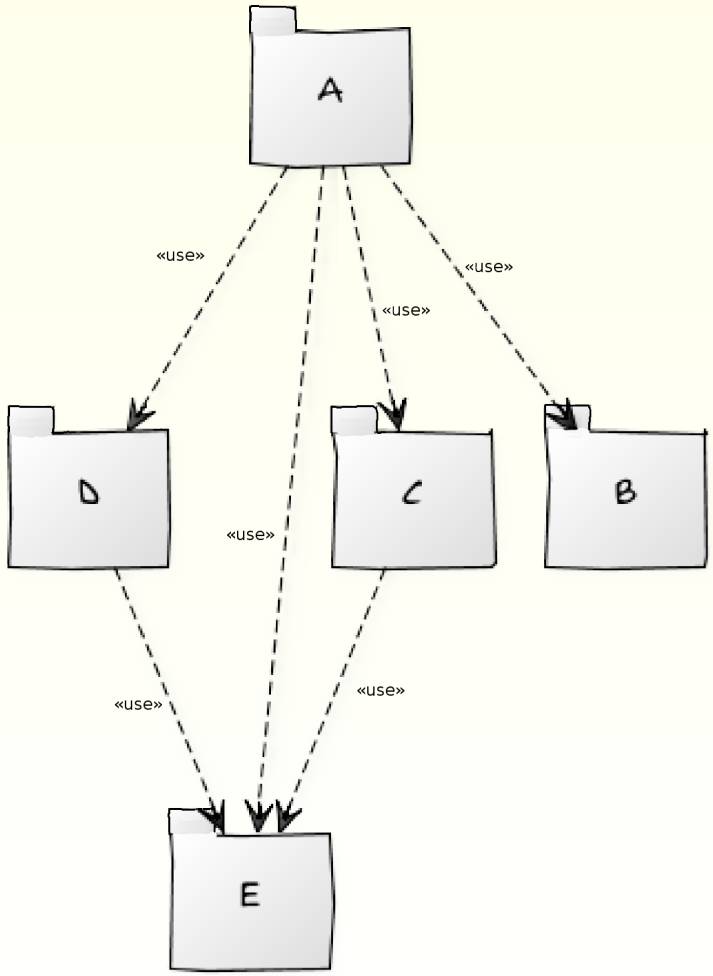
Architecture: My Definition

The software architecture describes the coarse structures of the software and defined how to *think* about it as a developer.

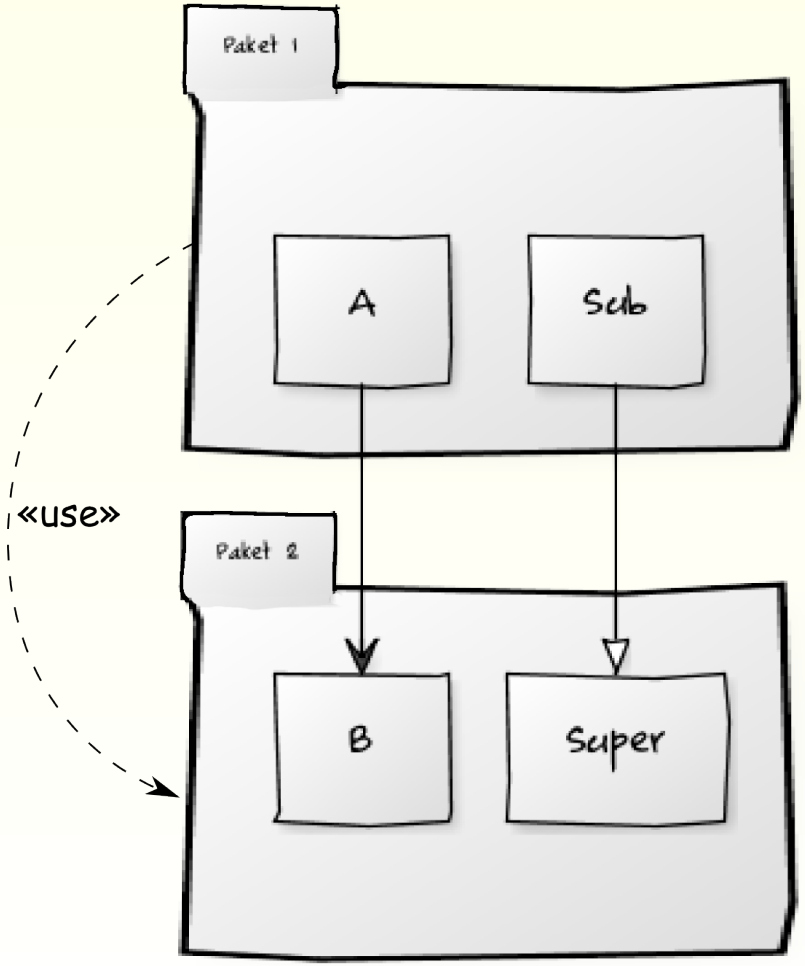
Coarse Structure

A coarse decomposition structure as a part of architecture

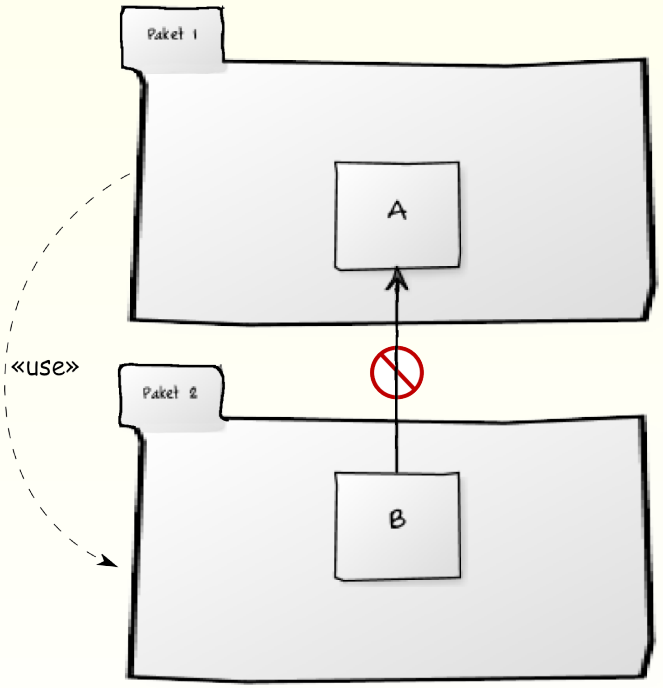
Constrain Communication (1/2)



Constrain Communication (2/2)

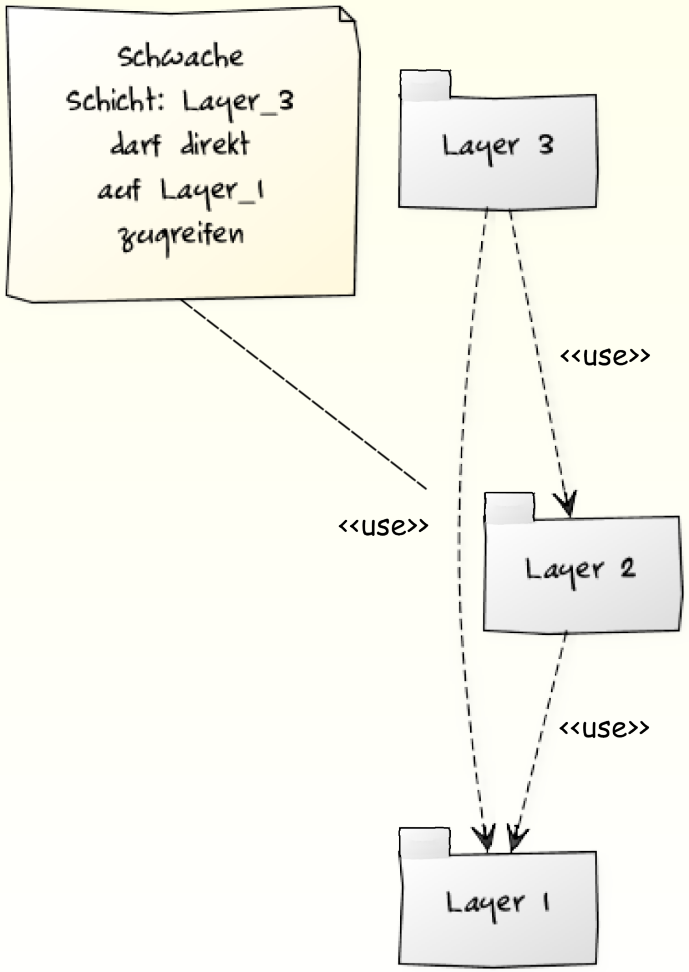


Dependency Inversion: Events

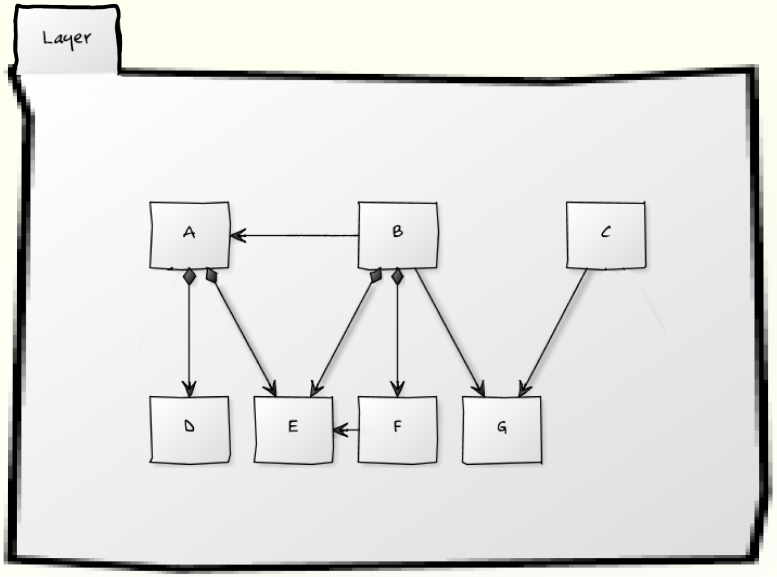


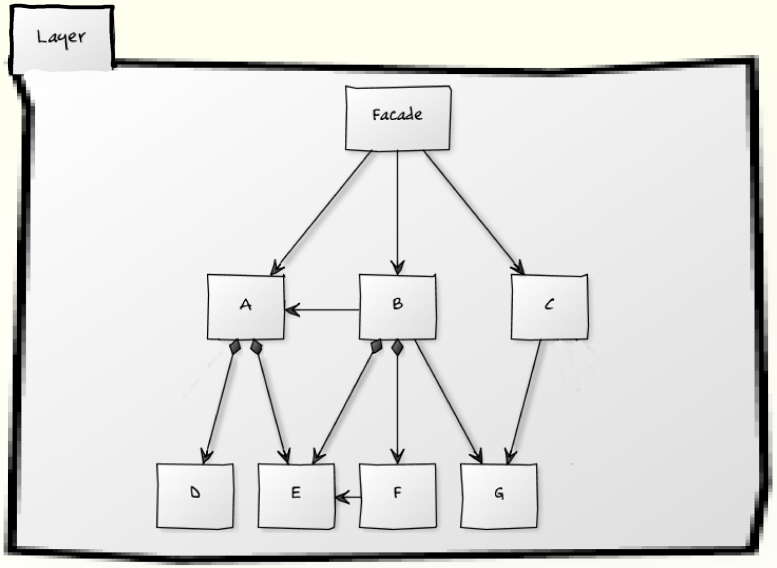

```
property OnSomeEvent: TNotifyEvent read
        FOnSomeEvent write FOnSomeEvent;
```

Layers


Schwache
Schicht: Layer_3
darf direkt
auf Layer_1
zugreifen
Layer 3
<<use>>
<<use>>
Layer 2
<<use>>
Layer 1

Layers and the Facade Pattern





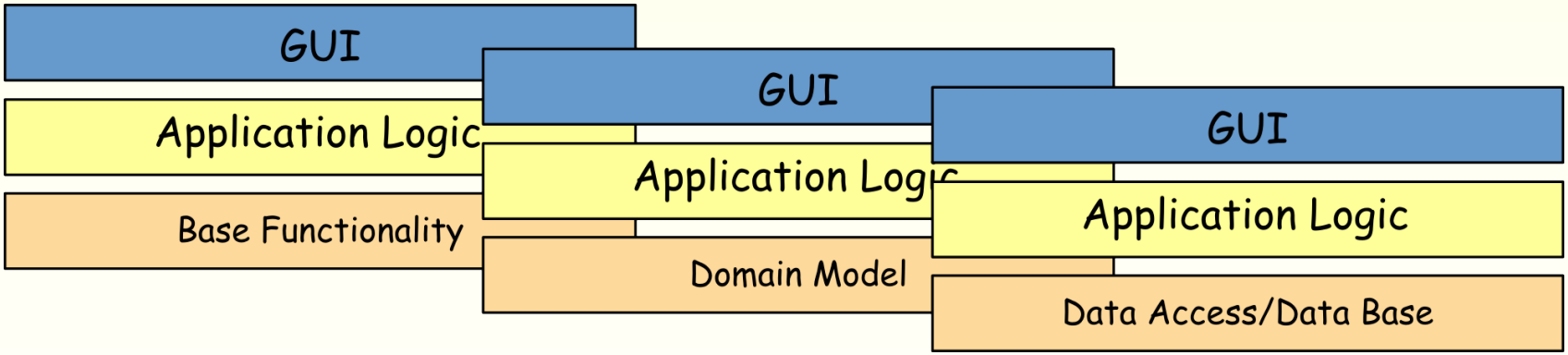
Layers Everywhere

- Networking:
  - Physical layer, link layer, network layer, transport layer, application layer
- APIs and Frameworks:
  - x86, WinAPI, RTL, VCL/FM
  - x86, WinAPI, CLR, .NET-Framework, SWF/WPF
- Typical Information Systems
  - GUI, application logic, data access
- ...

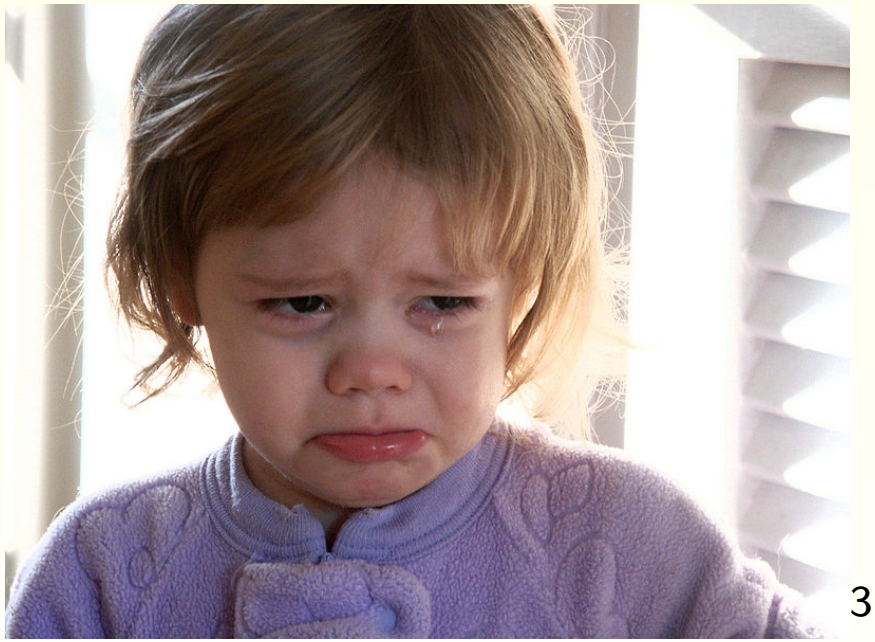
1, 2, 3, . . . , n Layers

- One layers: Everything is done in the form/every class may access every other $\Rightarrow$ chaos
- Two layers: e. g. form + data module
- Three layers: form + application logic + base layer (or similar)
- . . . Other breakdowns possible . . .

The Typical Three Layer Architecture



Tears? – Tiers!



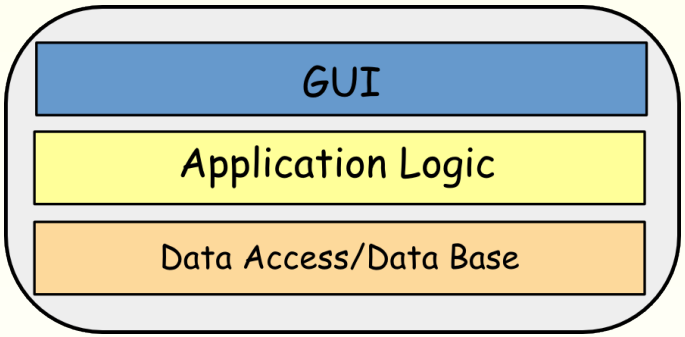
[3]

[3] CC-BY-SA 2.0 by Crimfants http://commons.wikimedia.org/wiki/File:Crying-girl.jpg

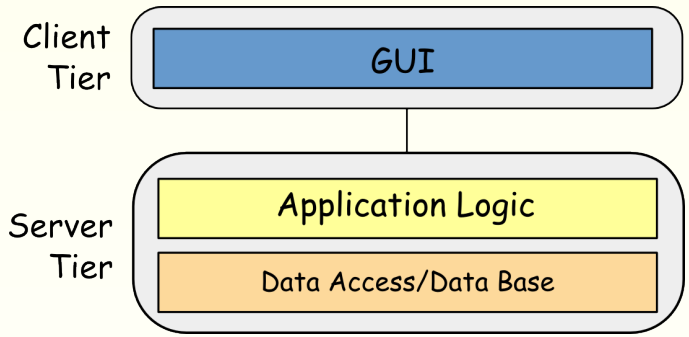
Layers vs. Tiers

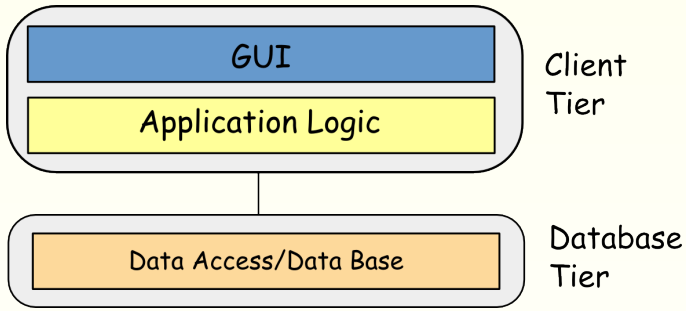
- Layer: logical separation
- Tier: physical separation

1-Tier

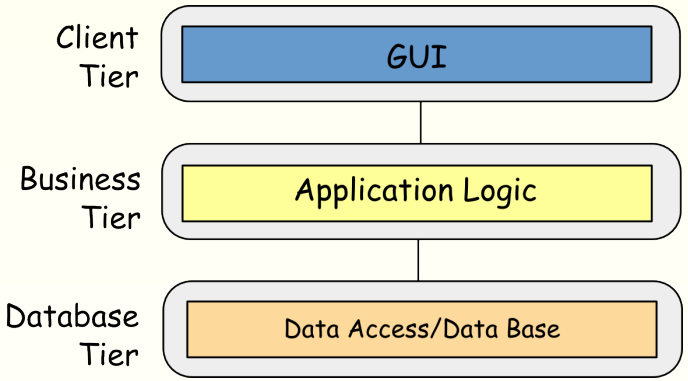


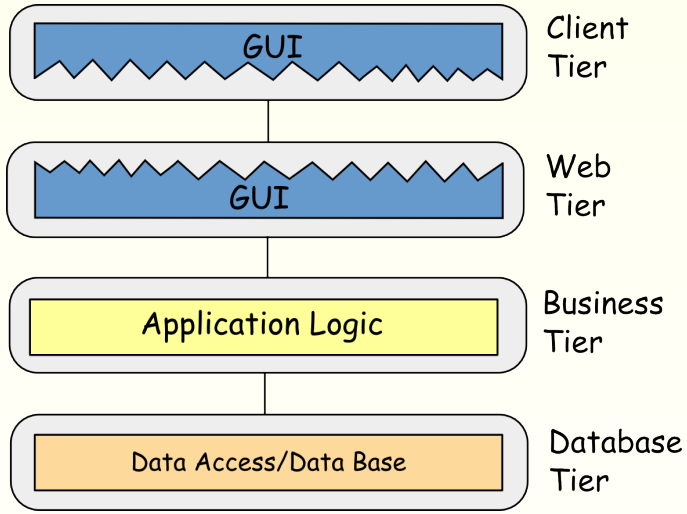
2-Tier


Client
Tier
GUI
Server
Tier
Application Logic
Data Access/Data Base


GUI
Application Logic
Client
Tier
Data Access/Data Base
Database
Tier

3-Tier, 4-Tier, n-Tier





Exceptions

Exceptional Cases

If only everything would be normal...

Types of Exceptional Cases

Avoidable, or not avoidable, — that is the question

Possibilities for Exception Handling

- Boolean return values
- Error codes
- Error states
- Error handlers
- Assertions
- Exceptions

Boolean Return Values

```
if OpenDialog.Execute then
begin
  ...
end;
```

Error Codes

```
const
  SHELLEXECUTE_MAX_ERROR = 32;

 ...

err := ShellExecute (...) ;
if err <= SHELLEXECUTE_MAX_ERROR then // something bad happened
begin
  case err of
    ERROR_FILE_NOT_FOUND: ...
    ERROR_PATH_NOT_FOUND: ...
    ERROR_BAD_FORMAT: ...
   else
      ...
  end;
end;
```

What's wrong here?

```
if ShellExecute (...) = ERROR_SUCCESS then
  ...
```

Error States

```
DoSomething(...);
 if  GetLastError <> NO_ERROR then
begin
    ...
end;
```

Error Handlers

Event: OnError

Assertions

```
procedure TSomeClass.DoSomething(param: TSomeObject);
begin
  Assert(param <> nil);
  ...
end;
```

Exceptions

```
procedure TMyList.Add(item: TMyItem);
begin
  if item = nil then
    raise EArgumentNil.Create('Cannot add nil to list.');
  ...
end;
```

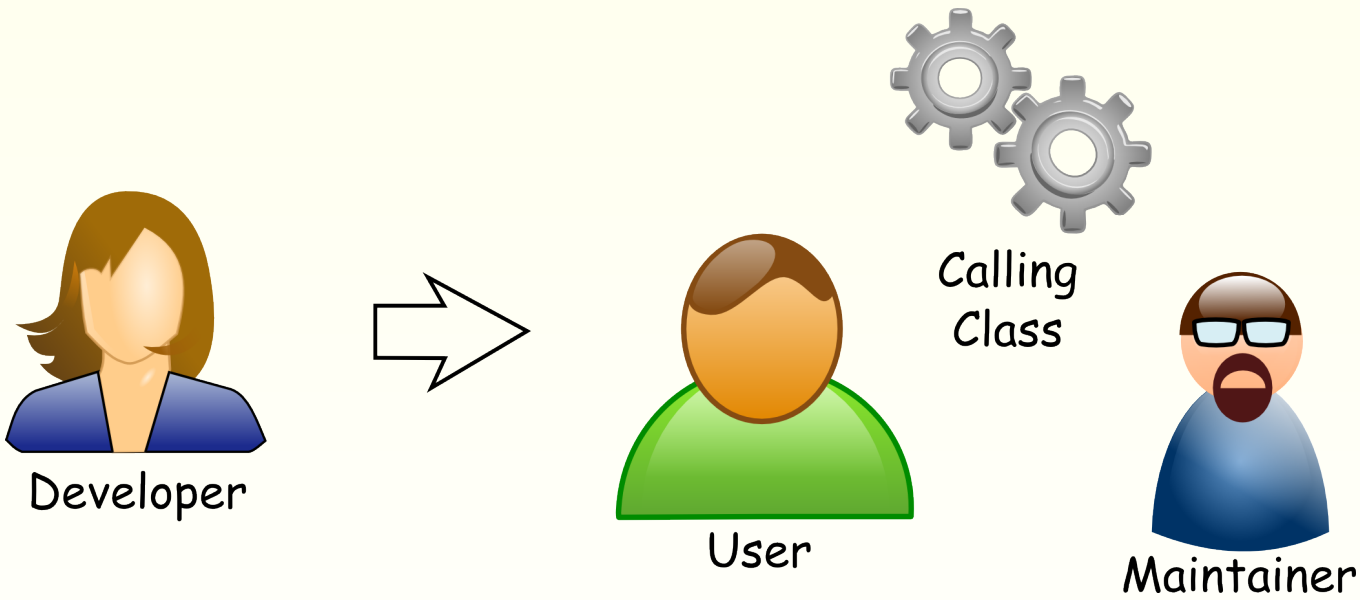
When to Use What?

- Boolean return values, error codes: if the exception is a regular part of the control flow (like `OpenDialog.Execute`)
- Error states: If the return value shall be used for other purposes
- Error handlers: for special cases
- Assertions: for uncovering bugs (i. e. avoidable exceptions)
- Exceptions: for everything else

How to Raise Exceptions


```
if coffeePot.isEmpty then
  raise EOutOfCoffee.Create('You drank too much coffee. Now there''s nothing left.');
```

Stakeholders: The Three Recipients



How to Catch Exceptions


```
var
  foo: TFoo;
begin
  foo := TFoo.Create;
  try
    try
      foo.Bar;
    except
      // handle Exception
    end
  finally
    foo.Free;
  end;
end;
```

How to Handle Exception (1/2)

- Avoidable Exceptions
  - Ignore them
  - Log them
  - Create a bug report
  - Exit program

How to Handle Exception (2/2)

- Unavoidable Exceptions: depends strongly on the concrete situation
  - Inform user
  - Rollback transaction
  - Retry
  - Reconnect
  - Remove client from the list
  - ...

Separating Normal Case and Exceptional Case (1/2)


```
if Bla(42) then
begin
  FillChar(param, SizeOf(param), 0);
  param.value := 21;
  o := Blubb(parem);
  if GetLastError = NoError then
  begin
    if o.DoSomething('not very interesting') <> SUCCESS then
      HandleDoSomethingFailling;
  end
  else
  begin
    LogError('Failure! ' + GetLastError);
    ShowMessage('something bad happened');
  end;
end
else
begin
  LogError('Failure in Bla!');
  ShowMessage('something bad happened');
end;
```

Separating Normal Case and Exceptional Case (2/2)


```
try
  Bla(42);
  FillChar(param, SizeOf(param), 0);
  param.value := 21;
  o := Blubb(parem);
  o.DoSomething('not very  interesting ');
except
  on e: EDoSomethingFailed do
  begin
    HandleDoSomethingFailling;
  end;
  on e: Exception do
  begin
    LogError('Fehler!' + e.Message);
    ShowMessage('something bad happened');
  end;
end;
```

Avoidable or Unavoidable?

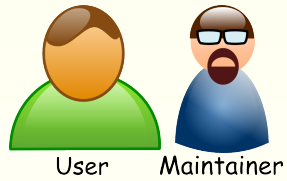
Calling Class

```
try
   ...
except
  on e: EAccessViolation do
  begin
     ...
  end;
end;
```

FileExists

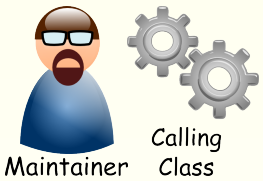

```
if  FileExists (someFile) then
begin
  LoadFile(someFile);
end;
```

Exception Message




```
raise EOutOfCoffee.Create('You are too thirsty, you idiot!');
```

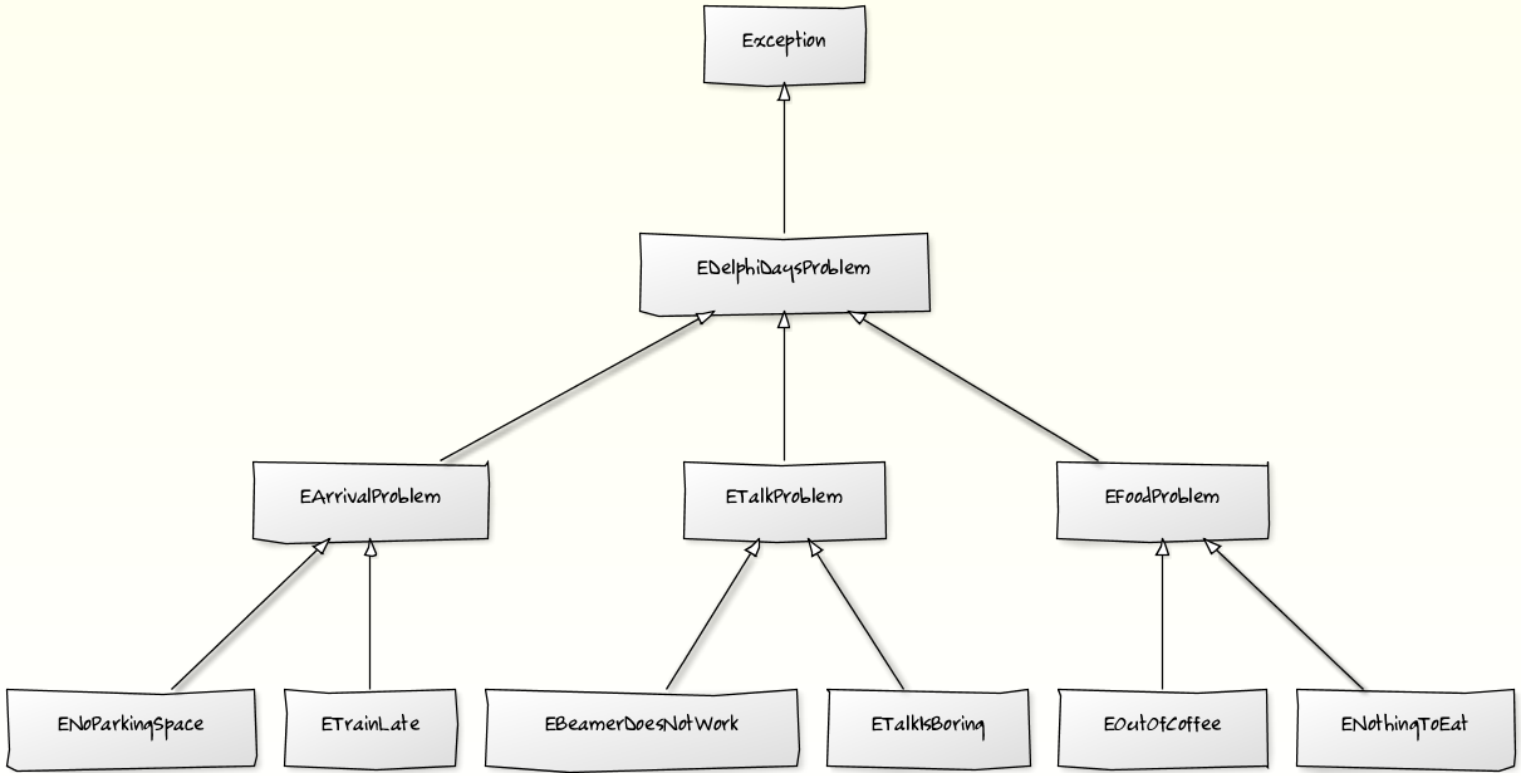
Exception Class




```
type
  EOutOfCoffee = class(Exception)
  end;
```

A Hierarchy of Exceptions

Calling Class



on-Statements

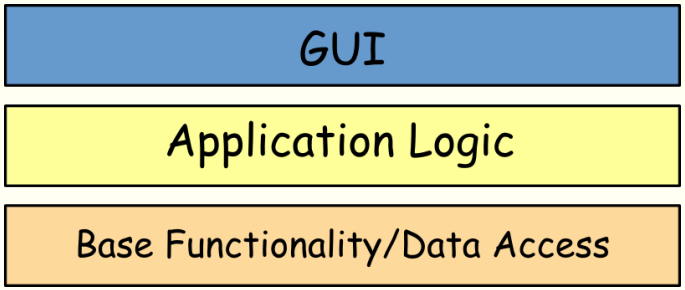
```
try
   ...
except
  on e: ETalkIsBoring do
  begin
     FallAsleep ;
  end;
  on e: ETalkProblem do
  begin
    ShakeHead;
  end;
  on e: EADUGProblem do
  begin
    Complain(e.Message);
  end;
end;
```

Existing Exception Classes

- `EAbort`
- `EArgumentException`
  - `EArgumentNilException`
  - `EArgumentOutOfRangeException`
- `EInvalidOpException`
- `ENoConstructException`
- `ENotImplemented`
- `ENotSupportedException`
- `EProgrammerNotFound`

Putting Everything Together

Exceptions in Layers



General Rule

General Rule about Exceptions in Layers

Catch Exceptions at the point where you know how to handle them. Not earlier and not later.

Rule of Thumb

Rule of Thumb

If in doubt, raise at the bottom and catch at the top.

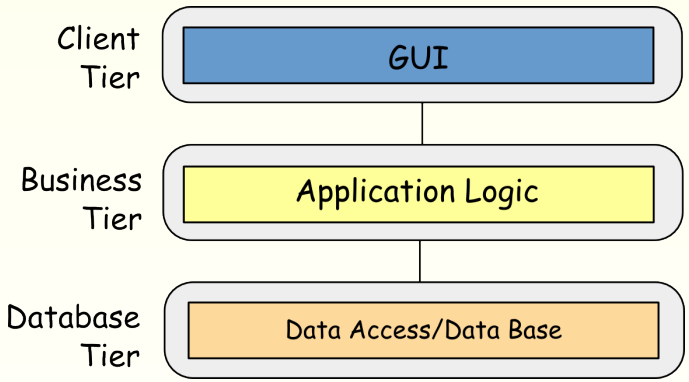
Problem

```
procedure TSettingsDialog.OKButtonClick(Sender: TObject);
begin
 try
  ...
 settingsObject . StoreSettings ;
except
on EFileStreamError do // ???
begin
  ...
end;
end;
end;
```

Exception Chaining

```
procedure TSettings.StoreSettings;
begin
try
 ...
except
on e: EFileStreamError do
begin
 raise ESettingsWriteError.Create('Could not write settings.', e);
end;
end;
end;
```

Exceptions and Tiers



Wrapper Classes

```
procedure TFooWrapper.DoSomething;
begin
  ret := NetworkCallToMethodDoSomethingInSomeFooObjectOnSomeOtherMachine;
  case ret of
    FOO_SUCCESS: // do nothing;
    FOO_WRITE_ERROR: raise EFooWriteError.Create('could not write...');
    FOO_READ_ERROR: raise EFooReadError.Create('could not read...');
  else
    raise EFooException.Create('Unknown Problem with Foo'); // base class or the Exceptions above
  end;
end;
```

Conclusion

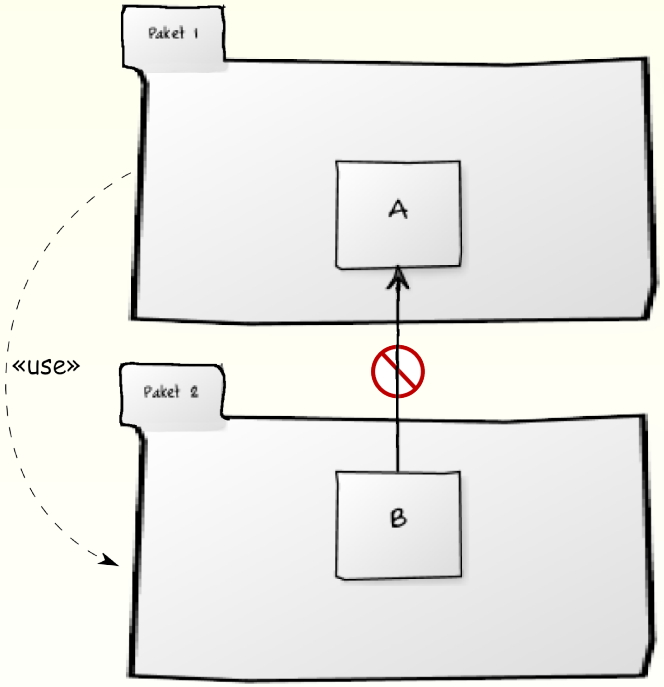
- Layers
  - Good architecture constrains the communication among the classes
  - Layers are a typical structure which supports this
  - Higher layers may access lower ones but *not* vice versa
- Exceptions
  - Separation of normal case and exceptional case
  - Avoidable and unavoidable exceptions
  - The three recipients of an exception
- Putting it all together
  - Exception chaining
  - Wrapper classes

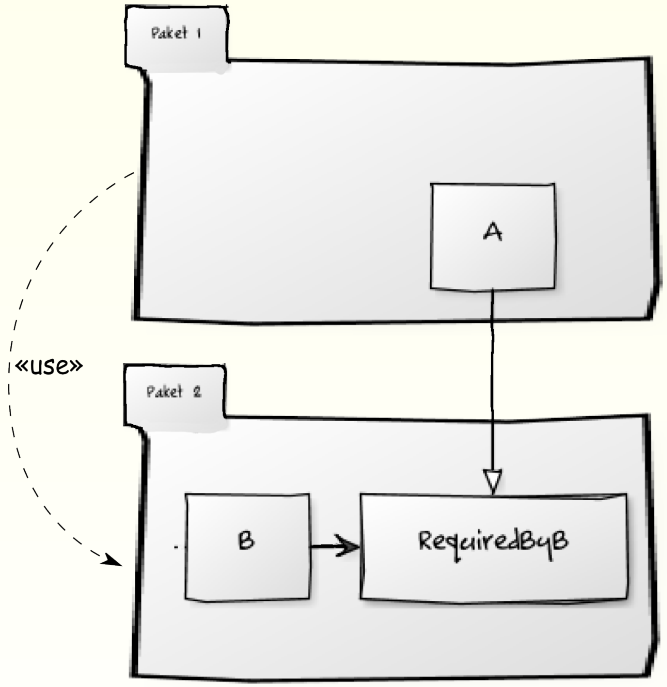
Thank You!

Questions?

Appendix

Dependency Inversion





Virtual Machines

Law of Leaky Abstractions

Law of Leaky Abstractions

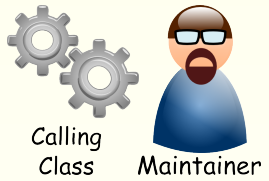
All non-trivial abstractions, to some degree, are leaky. [Spo02]

Guards


```
procedure AddItem(AItem: TMyItem);
begin
 if  AItem = nil then
 raise  EArgumentNil.Create('Cannot add nil.');

 ...
end;
```

Exceptions are Objects




```
EOutOfCoffee = class(Exception)
 private
 ...
 public
 property  NumberOfEmptyPots: Integer ...;
end;
```

References for Quotes


Len Bass, Paul Clemens, and Rick Kazman.
Software Architecture in Practice.
SEI Series in Software Engineering. Addison-Wesley, 2 edition, 2003.


Joel Spolsky.
The law of leaky abstractions.
`http://www.joelonsoftware.com/articles/LeakyAbstractions.html`, Nov 2002.

(real bibliography in talk notes)

Licence