AARHUS UNIVERSITET

Software Engineering and Architecture

Frameworks

AARHUS UNIVERSITET

Motivation

- I have presented some design principles
  - 3-1-2 process: *encapsulate variation, program to an interface and compose behavior using delegation*
  - Compositional design // SOLID principles
    - it is better to reuse code by delegation than inheritance
    - *Smaller, cohesive roles collaborating is better than a big blob*
  - Iterative and Incremental search for variability
    - supported by TDD
    - supported by refactoring

- ... and on the way we discovered some patterns

AARHUS UNIVERSITET

Abstracting

- What I have actually done is to show you the pieces that together form the puzzle called *frameworks:*
- **Framework:**
  - A framework is a set of cooperating classes that make up a reusable design for a specific class of software.

[GoF]

- Exercise:
  - Is the pay station a framework given this definition?
  - Argue why MiniDraw is a framework.

AARHUS UNIVERSITET

Framework Characteristics

- A framework is characterized by
  - reuse of *both* design as well as executable code
  - reuse within a well defined domain
    - Java Swing is for GUI's, not for banking…
  - high reuse percentage (70-90%)
  - must be customized for customer's particular need
- Examples:
  - Eclipse & IntelliJ IDE and Gradle plug-in architecture
  - Android / IOS
  - Web server frameworks (Jetty, Spark-Java, …)
  - HTML DOM and JavaScript
  - … MiniDraw

Other Definitions

- A framework is: a) a reusable design of an application or subsystem b) represented by a set of abstract classes and the way objects in these classes collaborate (Fraser et al. 1997).
- A framework is a set of classes that embodies an abstract design for solutions to a family of related problems (Johnson and Foote 1988).
- A framework is a set of cooperating classes that make up a reusable design for a specific class of software (Gamma et al. 1995, p. 26).
- A framework is the skeleton of an application that can be customized by an application developer (Fayad et al. 1999a, p. 3).
- A framework defines a high-level language with which applications within a domain are created through specialization (Pree 1999, p. 379).
- A framework is an architectural pattern that provides an extensible template for applications within a domain (Booch et al. 1999, p. 383).

AARHUS UNIVERSITET

Common aspects

- Skeleton / design / high-level language
  - … provide behavior on **high level of abstraction:** a design/skeleton/architecture
- Application/class of software
  - … has a **well defined domain** where it provides behavior
- Cooperating / collaborating classes
  - … define and limit **interaction patterns (protocol) between well defined roles**
- Customize/abstract classes/reusable/specialize
  - … can be **tailored** to a concrete context
- Classes/implementation/skeleton
  - ... is reuse of **code** as well as reuse of design

Variability points = Hotspots

AARHUS UNIVERSITET

Applications from frameworks

- A framework based application is composed of *three* portions of code
  - The framework code (external, third party)
  - The framework customization code (our own ‘tailoring’ code)
  - Non-framework related code (own domain code)

- Example: HotStone project
  - MiniDraw framework (basic 2D GUI control)
  - Customization code (adapt MiniDraw til HotStone graphics)
  - HotStone domain code (card draw, attacking, …)

AARHUS UNIVERSITET

Which leads to...

- Traditionally one distinguish between
  - Developers: Developing the software for end users
  - End users: Using the software
- Frameworks require one *additional* role
  - Framework developers: Develops the framework
  - Application developers: Adapt the FW for users
  - End users: Using the software

- I.e.: *The end users of a framework are themselves software developers!*

HotSpots…

”Separate code that changes from the code that doesn’t”

Definition: **Frozen spot**

A part of framework code that cannot be altered and defines the basic design and the object protocols in the final application.

Definition: **Hot spot**

A clearly defined part of the framework in which specialization code can alter or add behavior to the final application.

- Hot spots are also known as:
  - **Hooks / hook methods / callback functions**
  - **Variability points (our favorite term)**

Example

- The pay station system has **hotspots** regarding
  - receipt type
  - rate policy
  - display output
  - weekend determination
- but **frozen spots**, fixed behavior, concerning
  - coin types (payment options), numerical only display, only buy and cancel buttons, etc.
- Domain: Pay stations
  - no reuse in the electronic warfare domain ☺

AARHUS UNIVERSITET

Frozen spots

- Frozen is important : keep the code in the fridge!

> **Key Point: Frameworks are not customized by code modification**
>
> *A framework is a closed, blackbox, software component in which the source code must not be altered even if it is accessible. Customization must only take place through providing behavior in the hot spots by those mechanisms laid out by the framework developers.*

- Code modifications means unthawing and freezing again ☠ ☠ ☠

Why...

- I have realized that this point is so natural to me that I in early teaching simply has forgotten to emphasize it.
- Why
  - Consider Java Swing
    - If you had to *rewrite code in 8 different places in the Swing library to add a new special purpose visual component*
  - Then you had to
    - Understand the Swing code in deep detail (costs!)
    - *Reintroduce special purpose code* every time Sun/Oracle releases a new version of Swing (costs!!!)

Thus, I can get the Swing source code but I should *still* not change it!

Even More Why…

- Most of the time – you are of course not allowed to change the code !

> **Key Point: Frameworks are not customized by code modification**
>
> *A framework is a closed, blackbox, software component in which the source code must not be altered even if it is accessible. Customization must only take place through providing behavior in the hot spots by those mechanisms laid out by the framework developers.*

- Apple iOS
  - Is *closed-source* so you can of course not in any way change that
    - And if you could, all iPhone users had to install *your version of iOS* on their phone to use your great app… **That is not feasible, right?**

Morale

- You adapt a framework to your particular needs by
  - *Change by addition, not by modification!!!*
  - *Open for extension, closed for modification*

- You
  - Implement interfaces or extend existing classes
    - Concrete implementations, **real behavior filling out the responsibilities defined by the framework**
  - Tell the framework to use *your* implementations...

AARHUS UNIVERSITET

Which again leads to...

- If the framework has to use my concrete implementations then...

- It cannot itself create these objects
  - If MiniDraw itself defined
    - Drawing drawing = new CompositionalDrawing();
  - Then it was an application (fixed behavior), not a framework (adaptable behavior).

- Then ... How can it create objects?

Dependency Injection

Key Point: Frameworks must use dependency injection

Framework objects cannot themselves instantiate objects that define variability points, these have to be instantiated by the application code and injected into the framework.

- Typically an (OO) framework would use factory techniques like **abstract factory**
  - Or prototype or factory method patterns...
  - Or reading configuration files, or using conventions
    - Ruby on Rails, Grails, Maven, IntelliJ, Eclipse, use particular named files in specific directory paths to configure the frameworks
- *Or – by inheritance based techniques…*
  - *Android OS, LibGdx, and many others…*

Customization Techniques

AARHUS UNIVERSITET

Techniques to define hotspots

- We have object oriented composition to define hotspots. We have also looked at other techniques
  - Parameterization, overriding superclass methods

HotSpots

- Exercise: List techniques to *make the hotspots*
  - Hint: Consider techniques used in
    - MiniDraw
    - Swing
    - Collection classes
    - Eclipse
    - C library sorting algorithms
    - Intel i7 interrupt vector handling
    - Functional languages
    - Amiga libraries ☺
- Does a Framework require object-orientation?

AARHUS UNIVERSITET

A good framework...

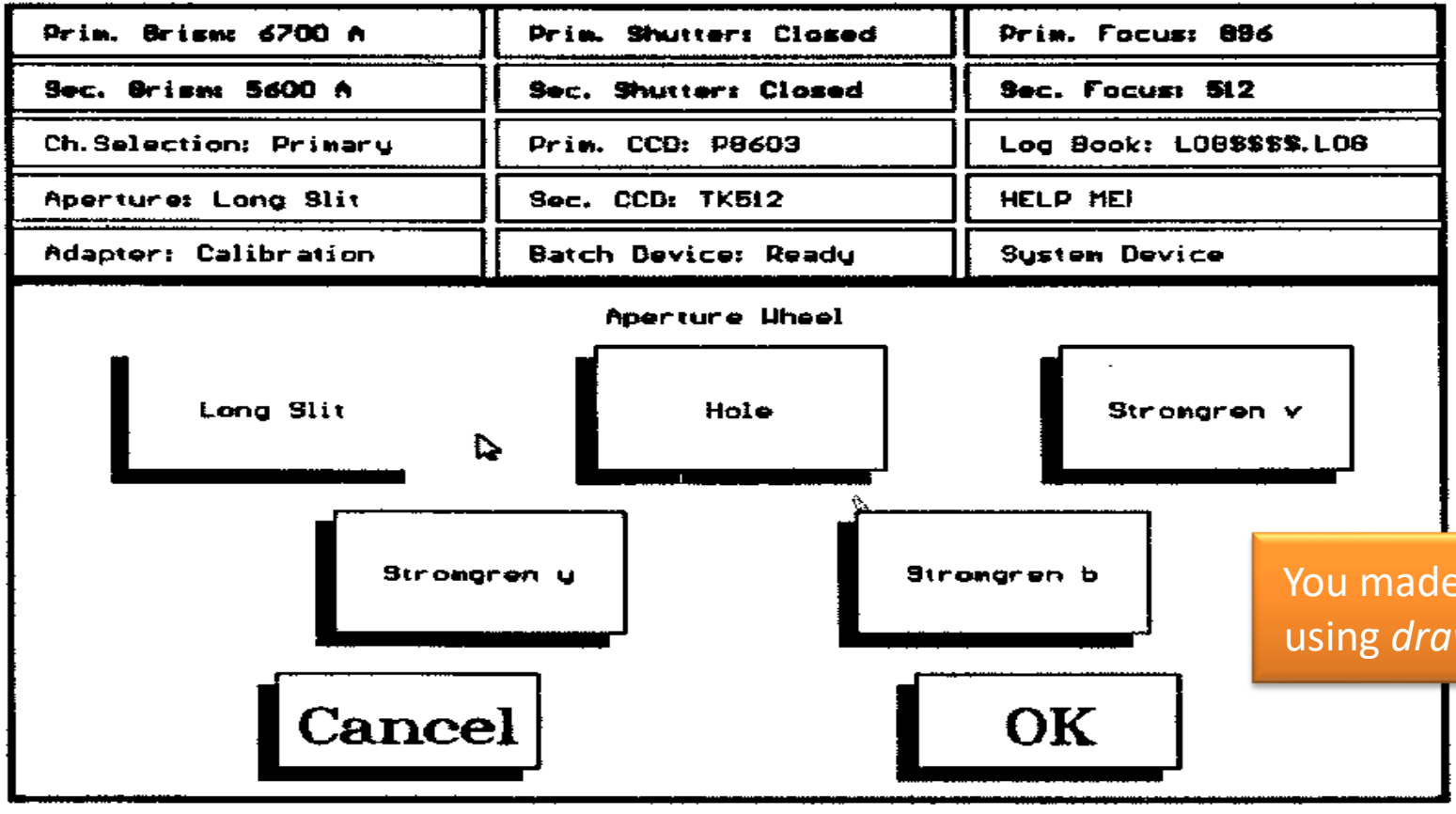
- ... Must give ”return on investment”
  - Investment: ”I have to learn how to use it”
  - Return: ”I get reliable software that does a lot”
- Android/Swing/MiniDraw/...
  - Consider writing the Android OS yourself!

Key Point: Frameworks should support the spectrum from no implementation (interface) over partial (abstract) to full (concrete) implementation for variability points

A framework provides the optimal range of possibilities for the application developer if all variability points are declared in interfaces, if these interfaces are partially implemented by abstract classes providing “common case” behavior, and if a set of concrete classes for common usages is provided.

AARHUS UNIVERSITET

In the old times...



You made the GUI using *drawLine !!!*

AARHUS UNIVERSITET

Framework Protocol

AARHUS UNIVERSITET

Design *and* Code Reuse

- "Cooperating / collaborating classes
  - … define and limit **interaction patterns (protocol) between well defined roles"**

- Frameworks require users (=developers) to understand the interaction patterns between objects in the FW and their own customization objects. That is, understand the **protocol.**

  - Ex: Understand the editor↔tool protocol in MiniDraw

AARHUS UNIVERSITET

Where is the protocol?

- So: The framework **dictates** the protocol!

- The question is: How?

Protocol

- The protocol arises because *objects in the framework invokes methods on objects that are defined by you.*

- These methods/functions/procedures that are predefined by the framework to be ‘overridable’ or customizable are the *hotspots.*

- A framework contains *frozen code* with embedded *hotspots* where your (unfrozen) code may be called.

AARHUS UNIVERSITET

Inversion of Control

- This property of frameworks is called

  - ***Inversion of Control***

- (”Hollywood principle: *do not call us, we will call you.)*

- *The framework dictates the protocol – the customization/hotspot code just has to obey it!*
  - “Do something when you are called”

AARHUS UNIVERSITET

Compare ‘traditional’ reuse

- Another type of reuse of code (more than design) is *libraries*

- I have never written my own *cosine* or *random* function – have you?
  - And a long time since I wrote a collection class, like List<T> ☺

- There are lot of libraries of usable behavior out there that does not *invert control.*

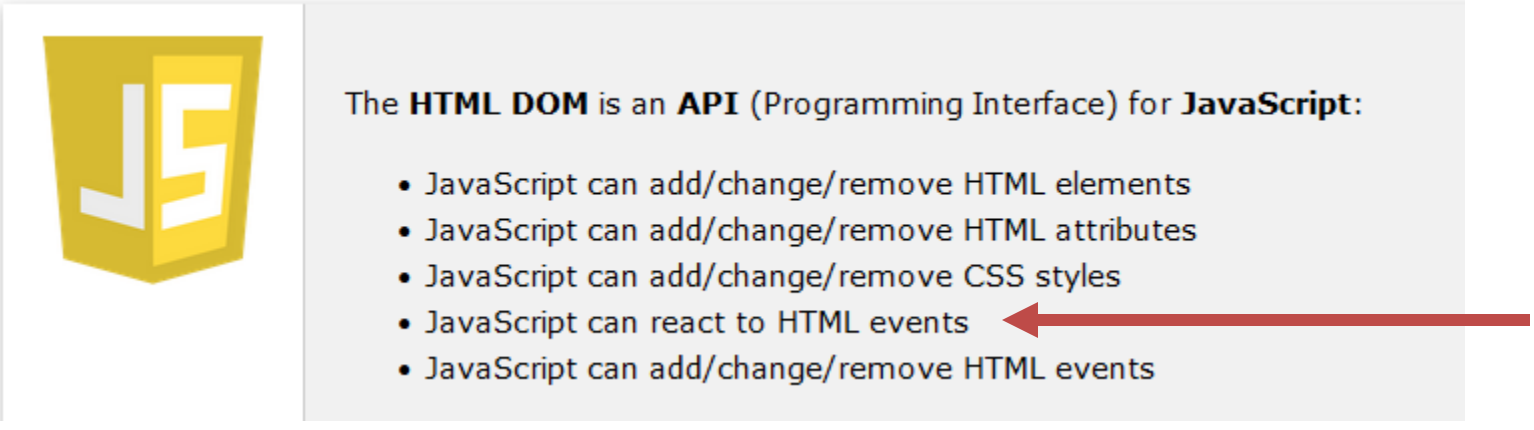
AARHUS UNIVERSITET

Framework Examples

AARHUS UNIVERSITET

HTML DOM and JavaScript

- The “operating system” of browsers
  - Event-Driven Programming of web pages



- The DOM is an **object model** of the page’s contents
  - In JavaScript you can associate *callback functions* with individual elements of that object

AARHUS UNIVERSITET

DOM and JavaScript

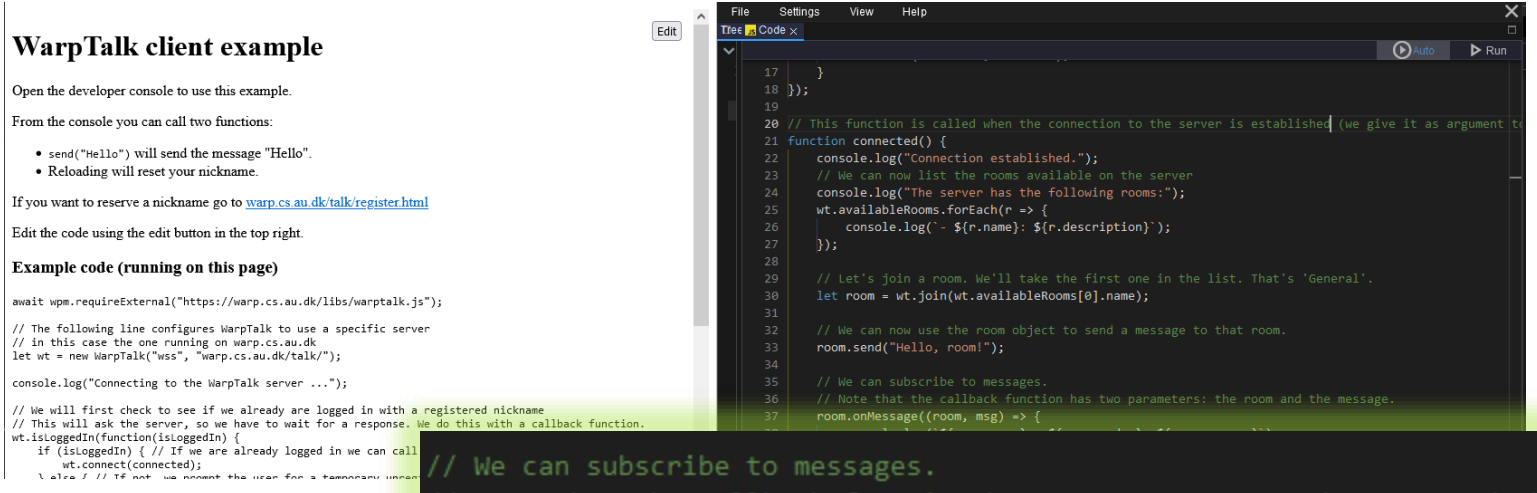
- The HTML DOM is a **Framework** in that each document element can be associated with **HotSpots** (event listeners). You can define these hotspots by **injecting** JavaScript functions as callback functions.

- The rendering of the DOM is a (lot of) **FrozenSpot.**

AARHUS UNIVERSITET

HCI Course

- WarpTalk Mandatory Exercise in HCI
  - *Disclaimer: I only spent 5 minutes and have not followed the course, so forgive if I am a bit of here ☺*
  - You fill in the hotspots, like **onMessage()** which allows you to adapt callback from server…



AARHUS UNIVERSITET

Examples: Android

```
package com.example.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

You do not call AndroidOS – *it will call you!*

Sr.No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

AARHUS UNIVERSITET

Examples: Android

```
package com.example.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

	Callback & Description
	lled when the activity is first created.
	he activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

You do not call AndroidOS – *it will call you!*

AARHUS UNIVERSITET

Example: Android GPS

- To get continuous tracking of your location, you can add an LocationListener (an Observer role)…

LocationListener

The LocationListener interface, which is part of the Android Locations API is used for receiving notifications from the **LocationManager** when the location has changed. The **LocationManager** class provides access to the systems location services. The LocationListener class needs to implement the following methods.

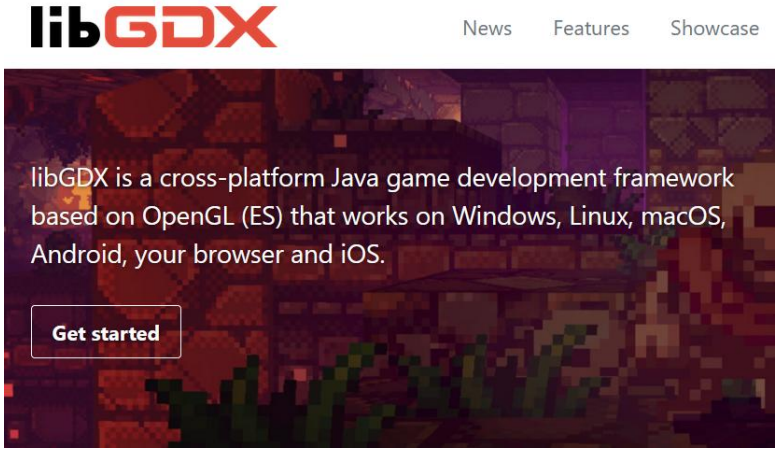
- **onLocationChanged(Location location)** : Called when the location has changed.
- **onProviderDisabled(String provider)** : Called when the provider is disabled by the user.
- **onProviderEnabled(String provider)** : Called when the provider is enabled by the user.
- **onStatusChanged(String provider, int status, Bundle extras)** : Called when the provider status changes.

AARHUS UNIVERSITET

Example: LibGDX

- Game Architecture:
  - loop at ~60 fps

```
import com.badlogic.gdx.Screen;
public class CrushRollerScreen implements Screen {
    @Override
    public void render() {
        batch.begin();
        batch.draw(bucketImage, bucket.x, bucket.y);
        for(Rectangle raindrop: raindrops) {
           batch.draw(dropImage, raindrop.x, raindrop.y);
        }
        batch.end();
```



- HotSpots for
  - Rendering, input handling, power management, sprite collisions,

Template Method

The central OO pattern to separate frozen and hot code

The core of the *inverted control*

Template Method

- Intent (Original GoF statement)
  - *Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.*

- Restatement
  - Some steps in the algorithm are fixed but some steps I would like to keep open for future modifications to the behavior

AARHUS UNIVERSITET

GoF Structure

- The structure in GoF is that of *subclassing*

```
abstract class AbstractClass {
  public void templateMethod() {
    [fixed code part 1]
    step1();
    [fixed code part 2]
    step2();
    [fixed code part 3]
  }
  protected abstract void step1();
  protected abstract void step2();
}
class ConcreteClass extends AbstractClass() {
  protected void step1() {
    [step 1 specific behavior]
  }
  protected void step2() {
    [step 2 specific behavior]
  }
}
```

AARHUS UNIVERSITET

The Multi-Dimensional Variability

- Subclassing handles multi-dimensional variability badly...

- Consider
  - Three variants of step1() required
  - Four variants of step2()
  - Any combination feasible

- How many subclasses?

```
abstract class AbstractClass {
  public void templateMethod() {
    [fixed code part 1]
    step1();
    [fixed code part 2]
    step2();
    [fixed code part 3]
  }
  protected abstract void step1();
  protected abstract void step2();
}
class ConcreteClass extends AbstractClass() {
  protected void step1() {
    [step 1 specific behavior]
  }
  protected void step2() {
    [step 2 specific behavior]
  }
}
```

Conclusion

- Conclusion:

Favor object composition over class inheritance

AARHUS UNIVERSITET

Exercise

- Rewrite the GoF structure to its compositional equivalent that is behavioral equivalent but follows the three *principles of flexible design.*

```
abstract class AbstractClass {
  public void templateMethod() {
    [fixed code part 1]
    step1();
    [fixed code part 2]
    step2();
    [fixed code part 3]
  }
  protected abstract void step1();
  protected abstract void step2();
}
class ConcreteClass extends AbstractClass() {
  protected void step1() {
    [step 1 specific behavior]
  }
  protected void step2() {
    [step 2 specific behavior]
  }
}
```

AARHUS UNIVERSITET

Interface Segregation Principle

- The ISP comes in handy
  - Separate each *cohesive* (subrole) into its own interface
  - HookInterface1 and HookInterface2

- Improved version:

```
class Class {
  private HookInterface1 delegate1; private HookInterface2 delegate2;
  public void setHook( HookInterface1 del1, HookInterface2 del2) {
    delegate1 = del1; delegate2 = del2;
  }
  public void templateMethod() {
    [fixed code part 1]
    delegate1.step1();
    [fixed code part 2]
    delegate2.step2();
    [fixed code part 3]
  }
}
interface HookInterface1 {
  public void step1();
}
interface HookInterface2 {
  public void step2();
}
class ConcreteHook1 implements HookInterface1() {
  public void step1() {
    [step 1 specific behavior]
  }
class ConcreteHook2 implements HookInterface2() {
  public void step2() {
    [step 2 specific behavior]
  }
}
```

AARHUS UNIVERSITET

Template Method Terminology

- These two variants have a name:
  - Original: subclassing + override hook methods
  - New: interface implementation + delegate to hooks

Definition: Unification

Both template and hook methods reside in the same class. The template method is concrete and invokes abstract hook methods that can be overridden in subclasses.

Definition: Separation

The template method is defined in one class and the hook methods are defined by one or several interfaces. The template method is concrete and delegates to implementations of the hook interface(s).

Exercise

- Identify template method in the pay station:

```
public void addPayment( int coinValue ) {
   switch ( coinValue ) {
   case 5:
   case 10:
   case 25: break;
   default:
     throw new IllegalCoinException("Invalid coin: "+coinValue+" cent.");
   }
   insertedSoFar += coinValue;
   _timeBought = rateStrategy.calculateTime(insertedSoFar);
 }
```

Exercise

- addPayment is template
  - A) coin validation; B) accumulation; C) time computation

```
public void addPayment( int coinValue ) {
  switch ( coinValue ) {
  case 5:
  case 10:
  case 25: break;
  default:
    throw new IllegalCoinException("Invalid coin: "+coinValue+" cent.");
  }
  insertedSoFar += coinValue;
  _timeBought = rateStrategy.calculateTime(insertedSoFar);
}
```

Frozen code 1

Frozen code 2

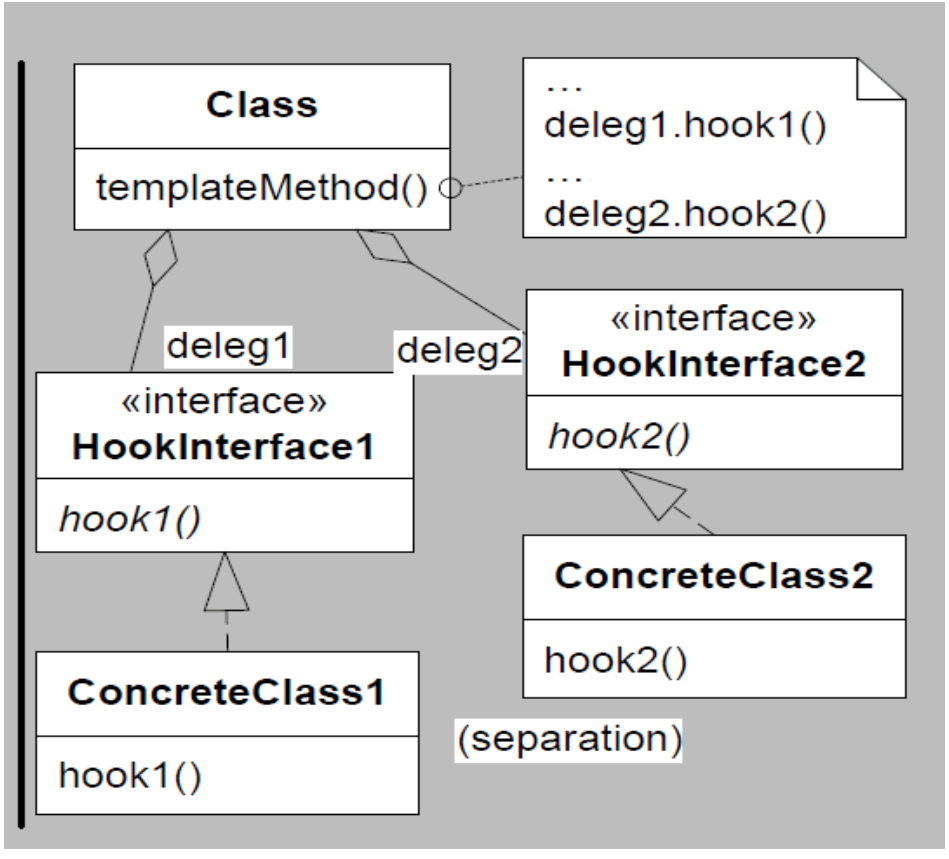
Hot code

Exercise

- How does *template method* relate to the *inversion of control* property of frameworks?

AARHUS UNIVERSITET

FRS Structure

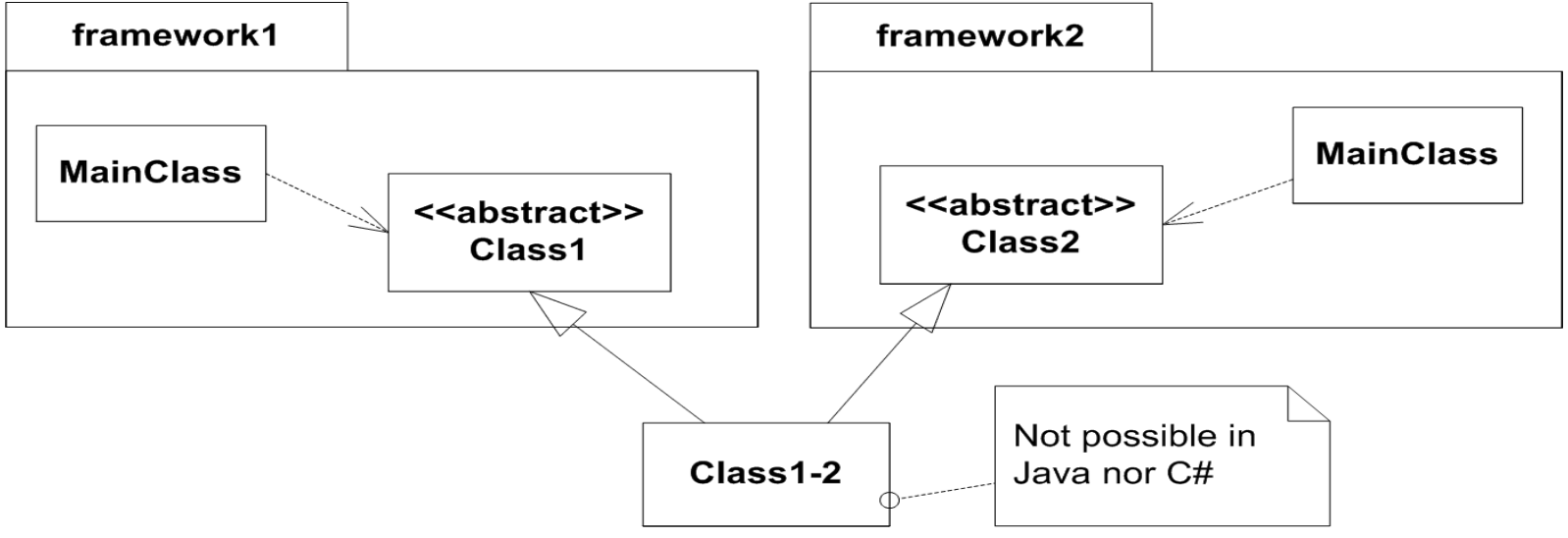


Another Case for Composition

AARHUS UNIVERSITET

Mixing Two Frameworks

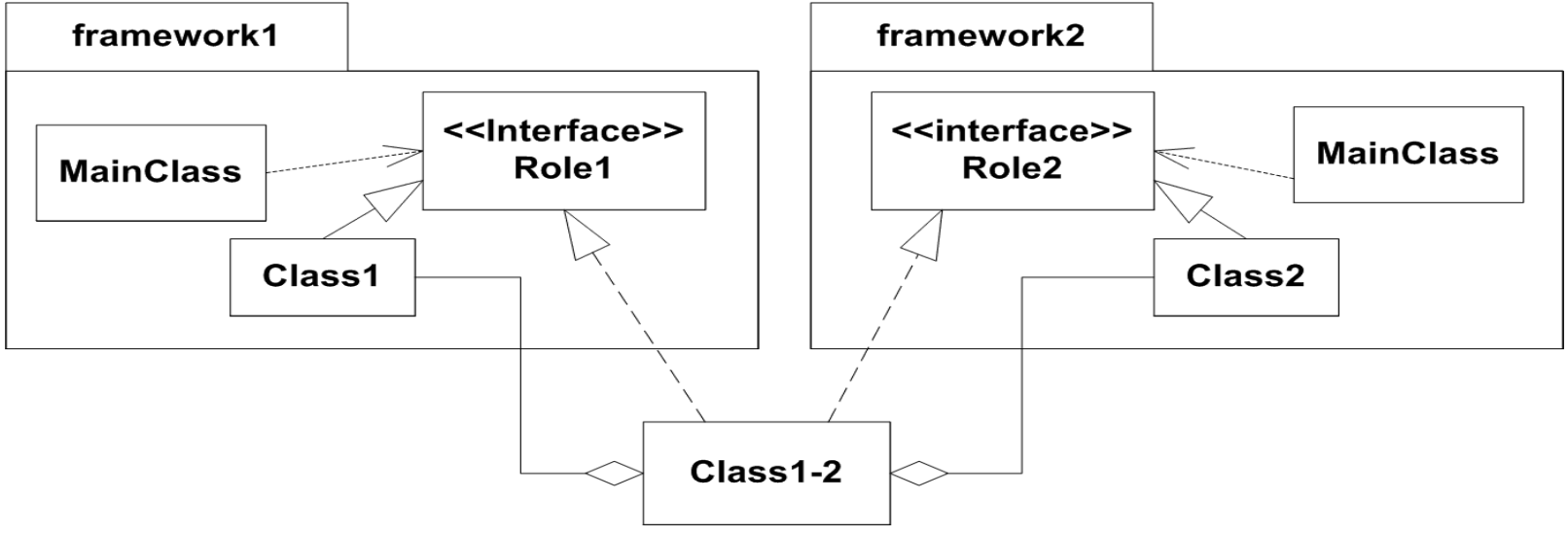
- Based upon the inheritance based template method



AARHUS UNIVERSITET

Mixing Two Frameworks

- But it does work using compositional template method



AARHUS UNIVERSITET

Example: HotStone GUI

- HotStoneDrawing implements Drawing, GameObserver