Informaticon

OOP
In PowerBuilder

April 2025

Motivation

Requirements always change

Motivation

Complexity explodes

Motivation

Maintenance turns into a nightmare

Informaticon

Simon Jutzi

DevOps Architect at Informaticon

- Various PBNI based libraries
- Automated PB migration to 2022R3
- Package manager for PowerBuilder
- CI/CD Pipeline for PowerBuilder applications

DevOps Architecture

Automating PB

Software Engineering

Golang / PowerBuilder / C++

IT Security

Cryptography enthusiast

Linux

RHEL / NixOS

Content

- Motivation
- OOP Toolbox
- Limitations in PowerBuilder
- Design Patterns
- Tips and tricks

Motivation

$$C(P) > C(½P) + C(½P)$$

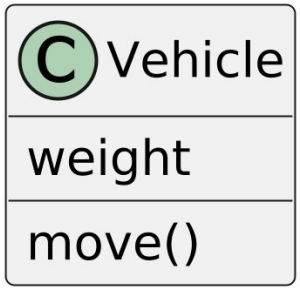
It is always easier (and cheaper) to create two small pieces rather than one big piece if the two small pieces do the same job as the single piece.

- Edward Yourdon, Larry L. Constantine 1975

OOP Toolbox

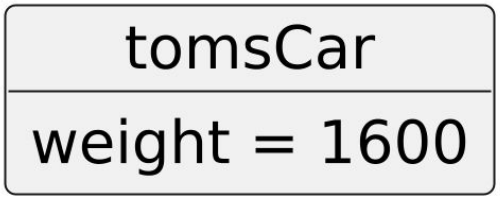
Class

- Blueprint for objects
- Defines behaviour and properties
- Is not a primitive data type
- Is stateless (no memory space*)



Object/Instance

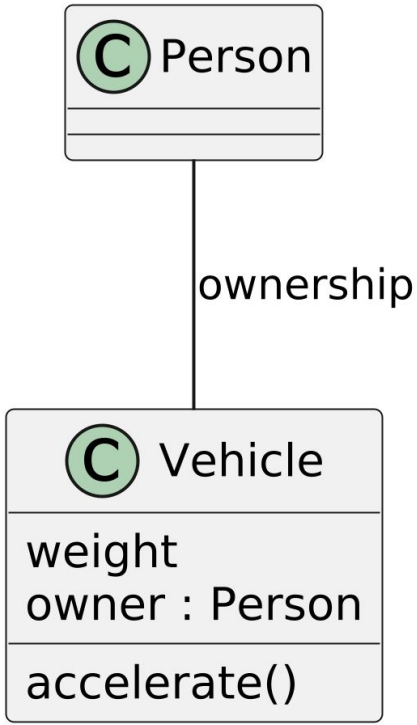
- Instance of a class
- Is stateful (instance properties)
- Has an identity



OOP Toolbox

Association

- (Weak) relationship between classes
- Model dependencies
- Usually needed for **instance variables**



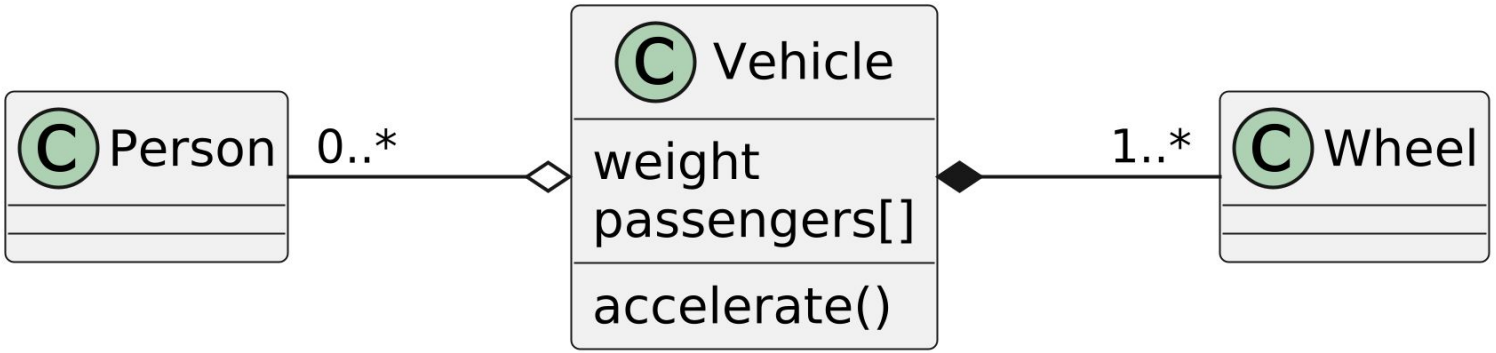
OOP Toolbox

Aggregation

- Weak relationship between classes
- Model components
- For **optional parts**
  (e.g. SQL-Drivers)

Composition

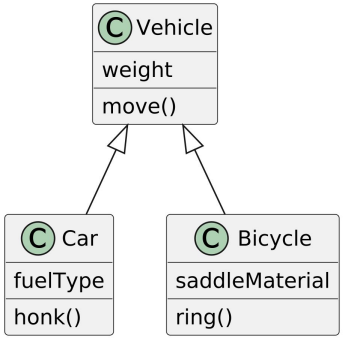
- Strong relationship between classes
- Model components
- Compositions live/die together
  (e.g. UI controls)



OOP Toolbox

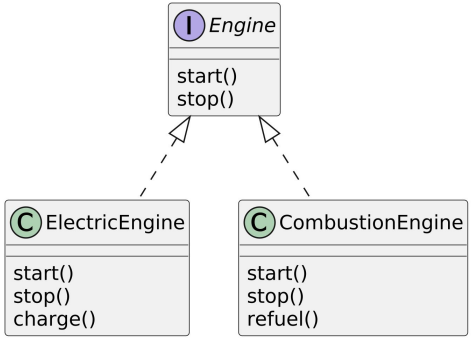
Inheritance

- Aka subtype polymorphism
- Derived class inherits variable and methods from base **class**
- **Define** common behaviour
- Re-use existing code



Interface implementation

- Aka subtype polymorphism
- Cass inherits variable and methods from **interface**
- **Declare** common behaviour



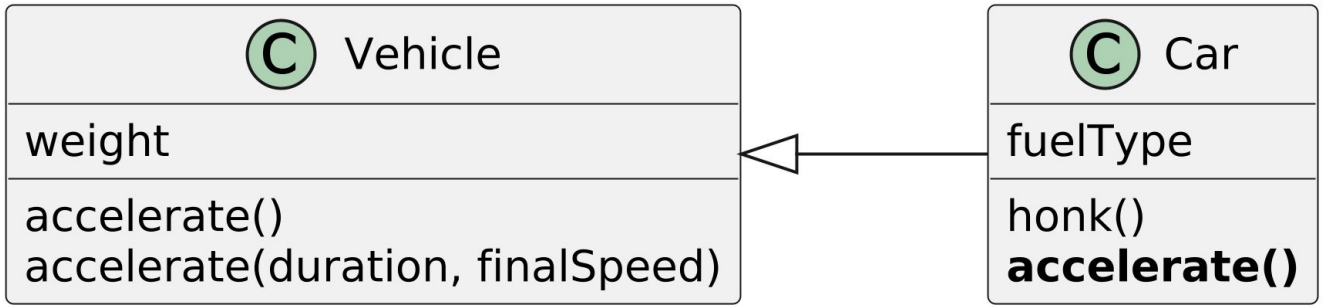
OOP Toolbox

Overloading

- Declare the same function/method with different arguments

Overriding

- Override method of base class

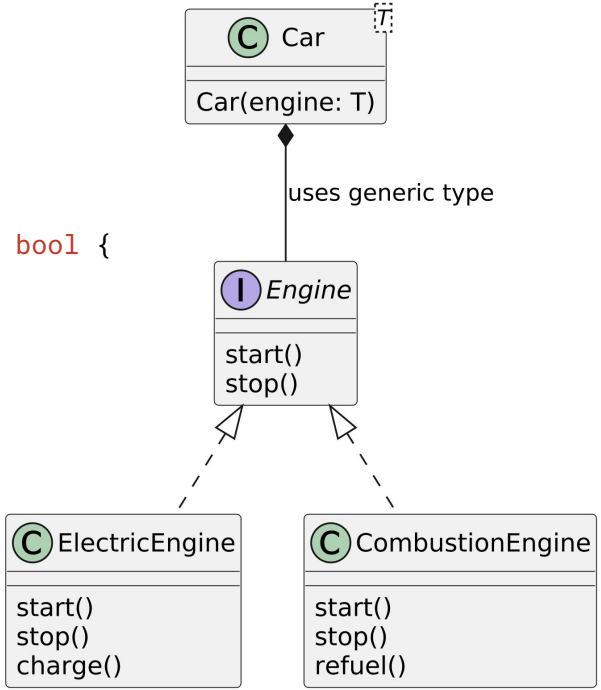


OOP Toolbox

Generics

- Aka parametric polymorphism
- Example in Go:

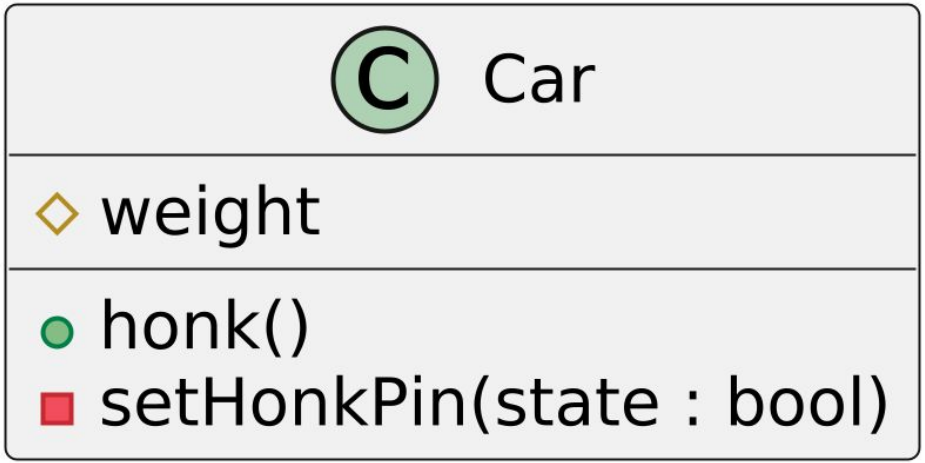
```go
func contains[T comparable](haystack []T, needle T) bool {
    for _, item := range haystack {
        if item == needle {
            return true
        }
    }
    return false
}

str := []string{"car", "bicycle", "motorbike"}
contains(str, "bicycle") // true

num := []int{3, 5, 7}
contains(num, 7) // true
```



OOP Toolbox

Encapsulation

- #Protected
- +Public
- -Private

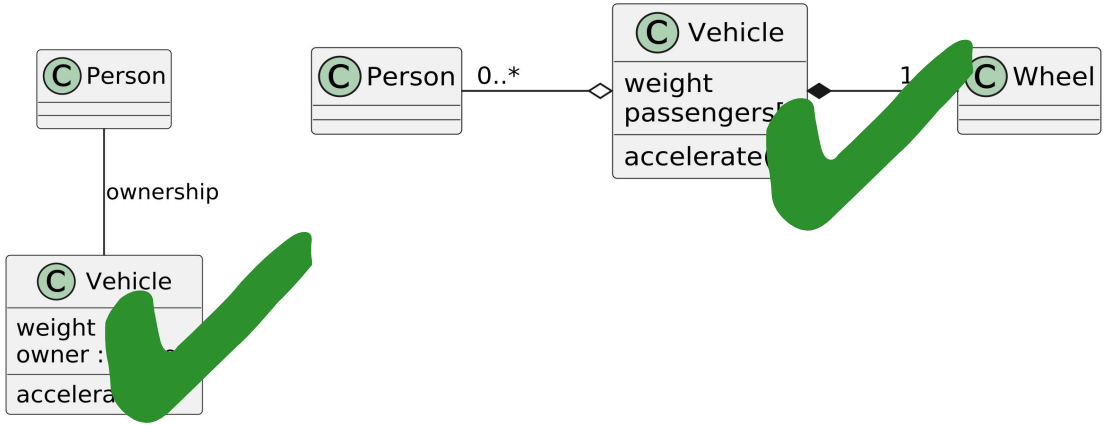


OOP Toolbox

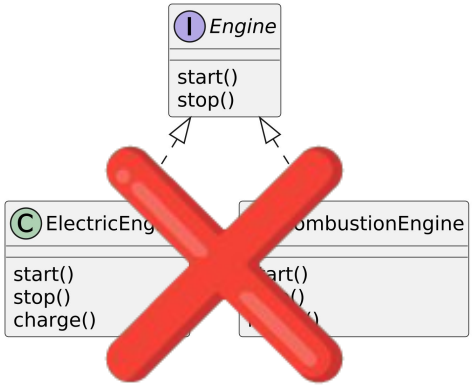
How and why is this relevant?

- Break things down into objects
- Provide well-defined APIs
- Simplify the needed mental model to understand the code
- Add plug-and-play functionality
- Reduces production and maintenance cost

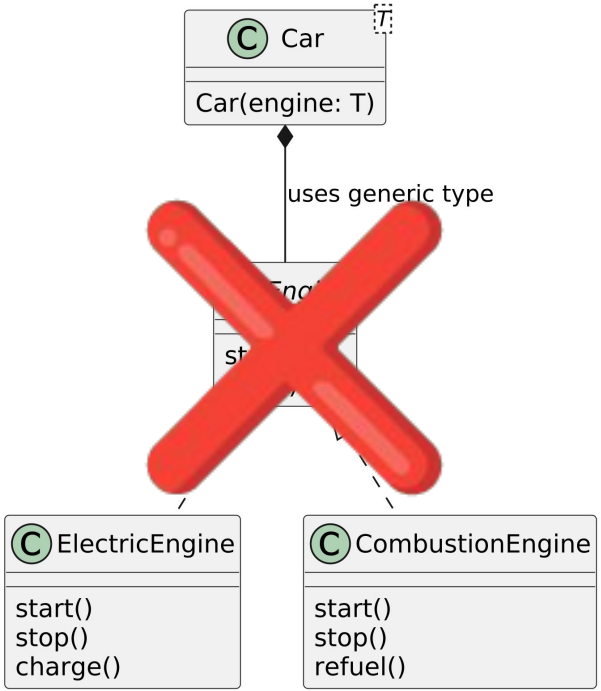
Limitations in PowerBuilder



Limitations in PowerBuilder

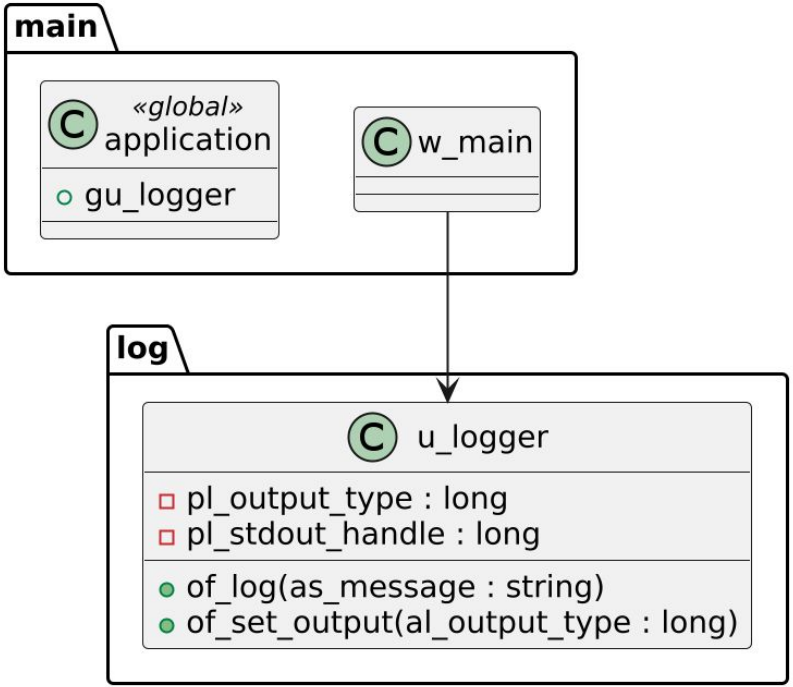


No constructor arguments



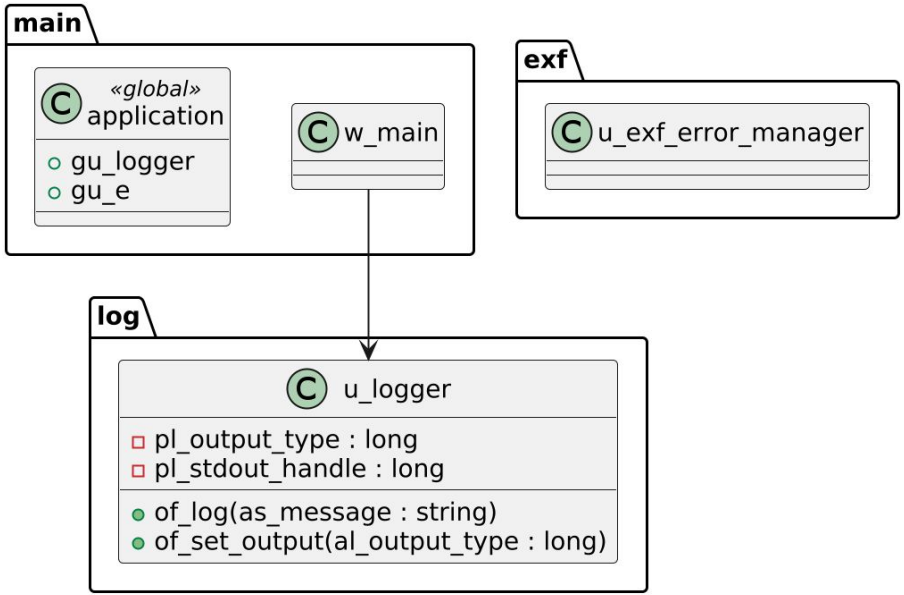
Let's write a logger library

Demo - Base



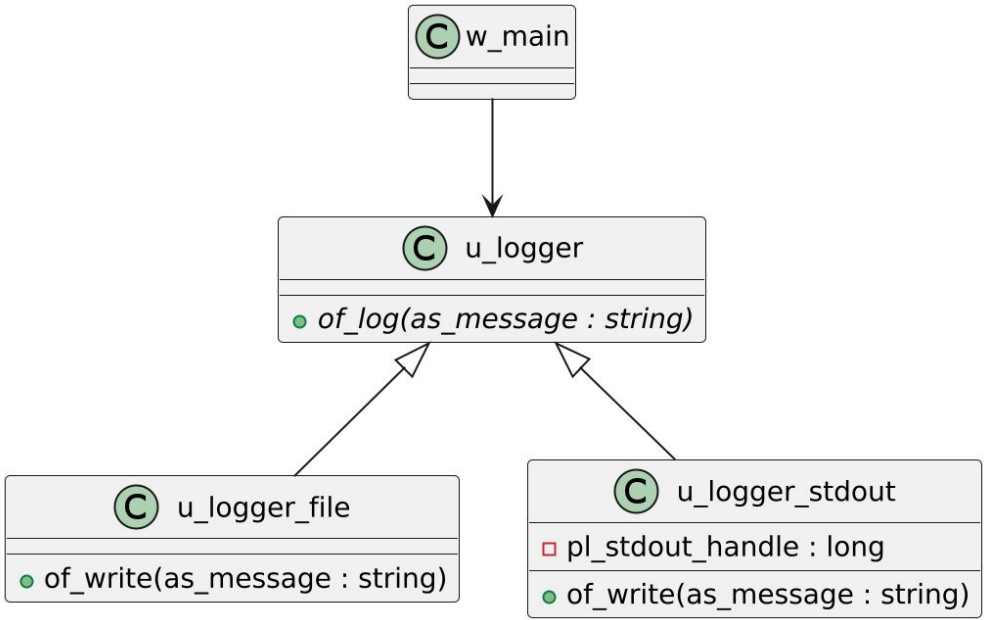
Let's write a logger library

Add Exception Handling



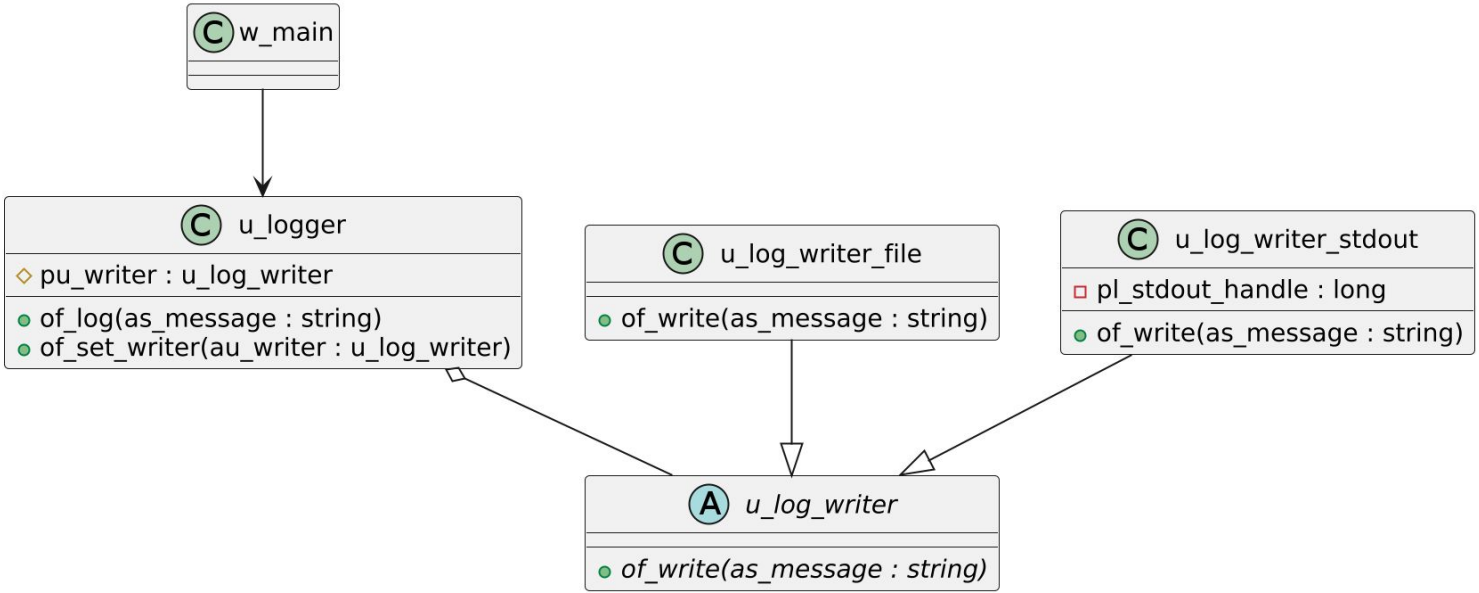
Let's write a logger library

Inheritance



Let's write a logger library

Aggregation



Let's write a logger library

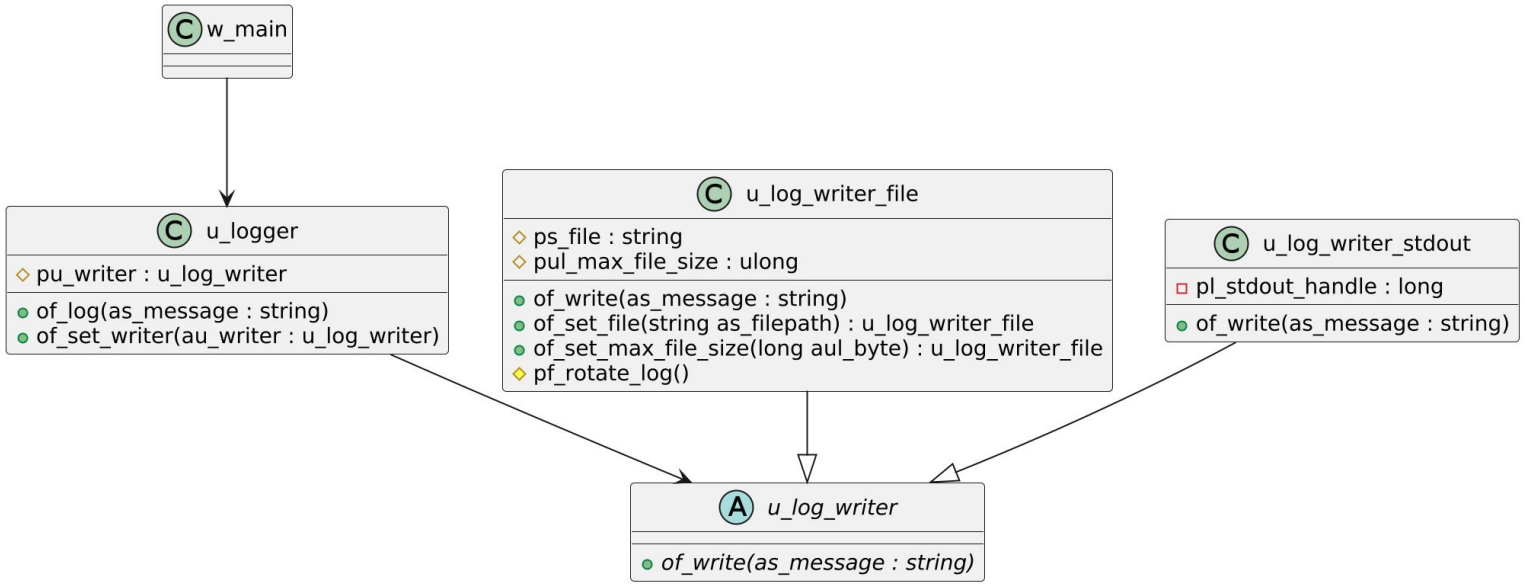
Singleton

- Use global function gf_get_logger

```
if not isvalid(u_logger) then
        u_logger = create u_logger
end if

return u_logger
```

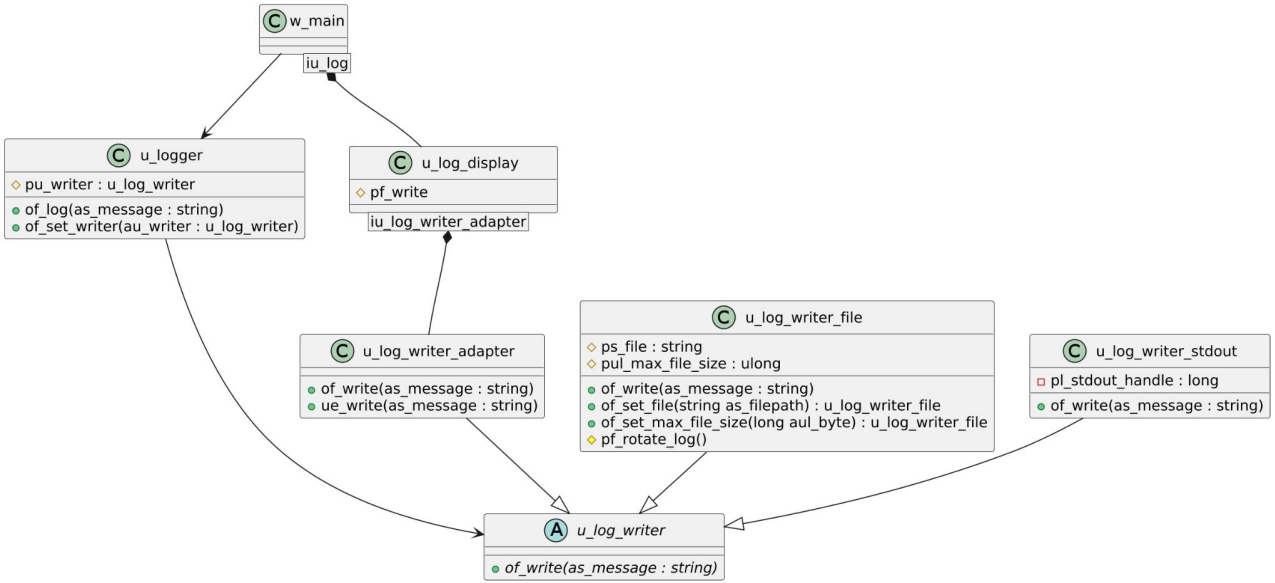
Let's write a logger library

Factory function



Let's write a logger library

Composition



Let's write a logger library

Dependency injection

- We already did it (u_log_writer)
- Practical example: Exception Framework

Tips and tricks

- Use private/protected and getter/setter functions
  - Prefix private/protected with a p
- Use descriptive names
  - Don't use lbo_disable_log=false ⇒ better: lbo_log_enabled=true
  - Use verb+object for function names (of_write_log, of_delete_file)
  - Use prefixes for classnames
- Simplify the creation of a mental model
  - Try to avoid state (instance variables)
  - Don't have unexpected side effects

Tips and tricks

- Use events only for...
  - User driven actions (UI related)
  - Callbacks
- Composition over Inheritance

Tips and tricks

Dynamic event call

- Demo: Test framework
  - u_tst_testcase.constructor

```
lcdf = this.classdefinition
lsdf = lcdf.scriptlist
for ll_i = 1 to upperbound(lsdf)
        if lsdf[ll_i].kind = scriptevent! and left(lsdf[ll_i].name, 3) = 'te_' then
                triggerevent(lsdf[ll_i].name)
        end if
next
```

Tips and tricks

Is A inherited from B?

```
//Argument 1: powerobject abo_object
//Argument 2: string      as_parent_classname
//Usage: of_is_inherited_from(u_log_writer_file, 'u_log_writer')

classdefinition lcd_temp
boolean lbo_null

if isnull(as_parent_classname) or not isvalid(apo_object) then
        setnull(lbo_null)
        return lbo_null
end if

lcd_temp = apo_object.classdefinition
as_parent_classname = lower(as_parent_classname)

do while isvalid(lcd_temp)
        if lower(lcd_temp.name) = as_parent_classname then
                return true
        end if
        lcd_temp = lcd_temp.ancestor
loop
return false
```

Tips and tricks

Quirks

- Garbage collector: mind the gap!
- PowerBuilder searches for objects from top to bottom
  - Duplicate class names are allowed
  - You can use this to maintain backward-compatibility
- PowerBuilder does not alway check class types correctly

```
3   u_log_writer lu_writer
4   lu_writer = create u_log_writer
5   of_test_type(lu_writer)
6   of_test_type()
```

of_test_type (**u_log_writer_file au_file**)

Informaticon

Thank you

github.com/informaticon/pb-design-patterns

linkedin.com/in/simonjutzi
simon.jutzi@informaticon.com

Questions?