Embedded C Programming Design Patterns

DESIGN PATTERNS TIL REN OG VEDLIGEHOLD BARE KODE FOR EMBEDDED SYSTEMER

Indhold

Hvad nu ?

Mål

- Gør det nemt at vedligeholde/øve kvalitet af driverlaget
  - Styre dependency
  - Indføre test
  - Mindske kompleksitet
- Øge kvaliteten af kode
  - Overskuelighed
  - Bedre dokumentation

Tiltag

- Dokumentation
  - Få beskrevet, hvad kravene til modulerne er
  - Lagt en overordnet struktur
- Øge overskuelighed
  - Styring af dependency
  - Ens formatering
  - Sørge for at hvis nøglepersoner forsvinder, så kan det rimeligt hurtigt forsættes af en anden
- Testbar
  - Moduler skal kunne testes individuelt
- Styring
  - Produkt owner

Object Pattern

- **Definition**
  - **Indhold sendes som parameter brug self parameter**
  - **Data er aldrig globalt**
  - **Funktioner har ikke statisk data**
  - **Data følger call path (ingen statik variabler)**

- **Eksempler**
  - **Grupper data i struct for at sortere dem**
  - **Object pattern er den primær måde at implemtere singleton**
  - **Abstrakt interface**
  - **Multi-thread design**
  - **Håndtere data i opaque pattern.**

- **Fordele**
  - **Ren interface til data**
  - **Re-entrant**
  - **Nem lock, da det er nemt at overskue data**
  - **Nem test via isolation**
  - **Data ligger i kode og gør at man ikke utilsigtet tilgår den.**

Implementation

"my_object.h"

```
struct my_object {
  uint32_t variable;
  uint32_t flags;
};

int my_object_init(struct my_object *self);
int my_object_deinit(struct my_object *self);
```

"my_object.c"

```
#include "my_object.h"
struct application {
     struct my_object obj;
}

int application_init(struct application *self){
  my_object_init(&self->obj);
}
```

Opaque Pattern

- Definition
  - Object definition i C-fil
  - Implementation giver størrelse
  - Bruger object pattern internt
  - Bruger ”New” og ”delete” funktioner
  - Bruger har kun pointer
- Eksempler
  - Isolere dependency
  - Kontrolleret adgang til data
- Fordele
  - Gem implementation
  - Begræns dependency

Implementation

H-File

```
struct opaque; // just a declaration
// init and deinit

int opaque_init(struct opaque *self);
int opaque_deinit(struct opaque *self);

// methods that operate on an opaque
void opaque_set_data(struct opaque *self, uint32_t data);
uint32_t opaque_get_data(struct opaque *self);
```

C-File

```
// actual definition of the struct in private space of the c file
struct opaque {
uint32_t data;
};

int opaque_init(struct opaque *self){
memset(self, 0, sizeof(*self)); /ZERO all
// do other initialization
return 0;
}

int opaque_deinit(struct opaque *self){
// free any internal resources and return to known state
self->data = 0;
return 0;
}

void opaque_set_data(struct opaque *self, uint32_t data){
self->data = data;
}

uint32_t opaque_get_data(struct opaque *self) {
return self->data;
}
```

Opgave

- Lav dette CAN modul om til Opaque Pattern?
- Hvordan kan man hængte en protokol på?

Singleton Pattern

- Definition
  - Control af multiple instance
  - Privat constructor
  - Stateless interface
- Eksempler
  - Logging
  - Configuration
  - Subsystem
  - Thread scheduler
- Fordele
  - Enkelt instans
  - Simple kode
  - Bedre performance
  - Håndtere shared resource

Implemetation

```
void main(void){
     struct object *self = object_acquire_singleton();
     if(self){
          object_do_something(self);
          object_do_something_else(self);
          object_release_singleton(&self);
     }
}
```

Factory Pattern

- Definition
  - Create object (som regel struct)
  - Manage memory pool
  - Konkrete implemtation
  - Bruger er som regel application kode og skjuler objecter fra bruger
- Eksempler
  - Init device driver
  - Memory pool og buffer
  - File system objecter
- Fordele
  - Øget fleksibilitet
  - Bedre kode organisation
  - Datadriven arkitektur
  - Bedre kode genbrug

Implemetation

Callback/Observer Pattern

- Definition
  - Object der opserveres (kigger med på data)
  - Callback receiver (den er modtager calls)
  - Separation mellem kalder og det kaldte
  - Exposing only a interface
- Eksempler
  - Notifikation
  - Observere state af object
  - One to many notifikation
  - Push-Subscribe pattern
- Fordele
  - Decoupling
  - En til mange
  - Reuse kode
  - Scalability
  - Let at vedligeholde

Implemetation

File: button.h

```
#include <stddef.h>

struct button {
      struct button_callback *cb;
};

// this is a public callback definition
struct button_callback {
      void (*cb)(struct button_callback *cb);
};

void button_init(struct button *self);
void button_deinit(struct button *self);
void button_add_callback(struct button *self, struct button_callback *cb);
void button_remove_callback(struct button *self, struct button_callback *cb);
void button_do_something(struct button *self);
```

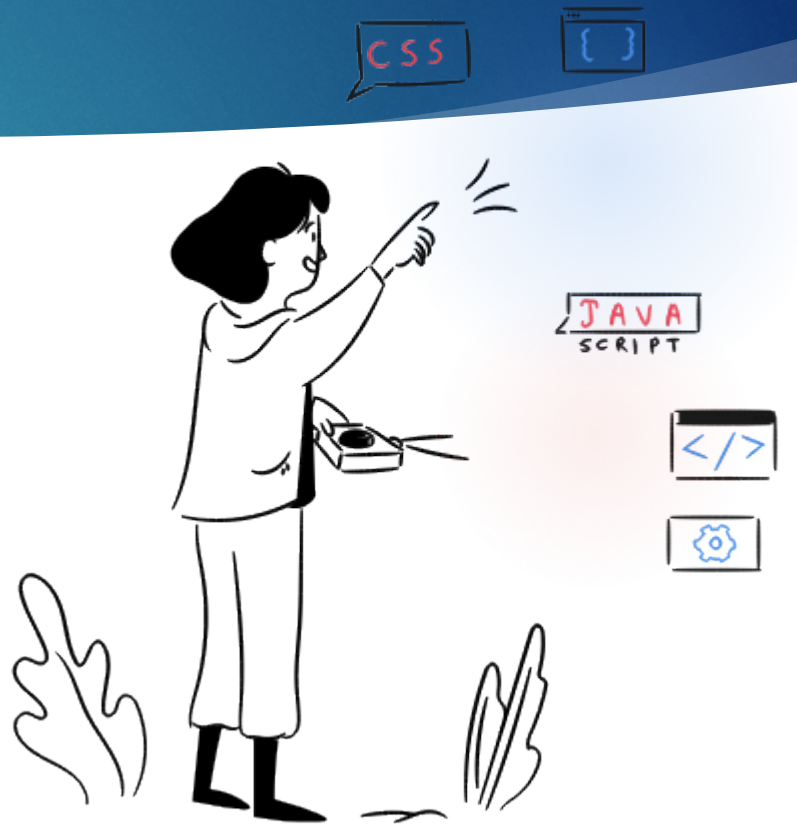
File: button.c

```
#include "button.h"
#include <string.h>

void button_init(struct button *s
      memset(self, 0, sizeof(*self)
}

void button_deinit(struct button
      self->cb = NULL;
}

void button_add_callback(struct b
      self->cb = cb;
}

void button_remove_callback(struc
      self->cb = NULL;
}

void button_do_something(struct button *self){
  if(self->cb)
    self->cb->cb(self->cb); // call the callback
}
```

```
struct button btn;

button_init(&btn);
button_add_callback(addTotal)
button_do_something(&btn);
button_deinit(&btn);
```

Opgave

- Add debouncing
- Add Press/release event
- Add LongHold



Inheritance Pattern

- Definition
- Eksempler
  - Kode genbrug
  - Generiske data struktur
  - Extend andre object med nye feature
- Fordele
  - Clean design gennem hierarki design
  - Mere simple kode, ved at samle kode i hierakisk orden

Implemetation

```
struct my_object {
  struct base_one base_one;
  struct base_two base_two;
};
```

Virtual API Pattern

- Definition
  - Organisere objecter, der er froskellige, men opfører sig på samme måde
  - Ligner Abstract interface i c++
- Eksempler
  - Device Drivers. Uart kan være forskellige, men vi laver et fælles interface
  - Plugin f.eks. dynamisk libery
  - Abstract data type
- Fordele
  - Decoupling
  - Type safety. Ingen void*.
  - Opaque handles. Abstract API

Implemetation

Return Value Pattern

- Definition
- Eksempler
- Fordele

Implemetation

Concurrency Pattern

- Definition
- Eksempler
- Fordele

Implemetation

Typer af designpattern

- Obejct Pattern: Gruppere data i class og lav construktor og destruktor
- Opaque Pattern: Gør implementationen privat
- Singleton Pattern: Single instance og global access
- Factory Pattern: Yderelig abstraktion
- Callback Pattern: Styr callback mellem instance
- Inheritance Pattern: Nedavle objecter
- Virtual API Pattern: ”smart” polymorphism
- Brigde Pattern