Noter til "GoF Template Method, GoF Strategy"

2. Agenda

- GoF Template Method
- GoF Strategy
- Sammenligning
- Laboratorieøvelse: “SuperSorter”

3. GoF Template Method

- **Intent:** Definer skellettet af en algoritme i en operation, idet nogle trin udskydes til underklasser.
- **Struktur:**
  - **AbstractClass:**

```
class AbstractClass {
public:
    void TemplateMethod() {
        MethodX();
        MethodA();
        MethodB();
        MethodY();
    }
protected:
    virtual void MethodA() = 0;
    virtual void MethodB() = 0;
private:
    void MethodX() { std::cout << "MethodX called\n"; }
    void MethodY() { std::cout << "MethodY called\n"; }
};
```

  - **ConcreteClass:**

```
class ConcreteClass1 : public AbstractClass {
protected:
    void MethodA() override { std::cout << "ConcreteClass1 MethodA
called\n"; }
    void MethodB() override { std::cout << "ConcreteClass1 MethodB
called\n"; }
};
```

4. GoF Strategy

- **Intent:** Definer en familie af algoritmer, kapsl hver af dem ind, og gør dem udskiftelige ved kørselstidspunkt.
- **Struktur:**
  - **Strategy Interface:**

```
class IStrategy {
public:
    virtual void Execute() = 0;
};
```

  - **Concrete Strategies:**

```
class ConcreteStrategyA : public IStrategy {
public:
    void Execute() override { std::cout << "ConcreteStrategyA
executed\n"; }
};
class ConcreteStrategyB : public IStrategy {
public:
    void Execute() override { std::cout << "ConcreteStrategyB
executed\n"; }
};
```

  - **Context:**

```
class Context {
private:
    std::shared_ptr<IStrategy> strategy;
public:
    void SetStrategy(std::shared_ptr<IStrategy> strategy) { this-
>strategy = strategy; }
    void ExecuteStrategy() { strategy->Execute(); }
};
```

5. Sammenligning af Template Method og Strategy

- **Template Method:** Bruger arv, hvor adfærd er fast ved kompileringstid.
- **Strategy:** Bruger delegation, hvor adfærd kan ændres ved kørselstidspunkt

Noter til "GoF Factory Method, GoF Abstract Factory"

1. Introduktion til Designmønstre

- Forelæser: Claudio Gomes
- Version: 1.0.2

2. Agenda

- GoF Factory Method
- GoF Abstract Factory

3. GoF Factory Method

- **Intent:** Definer en grænseflade til at skabe et objekt, men lad klasserne, der implementerer grænsefladen, bestemme, hvilket objekt der skal instantieres.
- **Struktur:**
  - **Product Interface:**

```
class Product {
public:
    virtual void Use() = 0;
};
```

  - **Concrete Products:**

```
class ConcreteProductA : public Product {
public:
    void Use() override { std::cout << "Using ConcreteProductA\n"; }
};
class ConcreteProductB : public Product {
public:
    void Use() override { std::cout << "Using ConcreteProductB\n"; }
};
```

- **Creator:**

```
class Creator {
public:
    virtual std::shared_ptr<Product> FactoryMethod() = 0;
    void AnOperation() {
        auto product = FactoryMethod();
        product->Use();
    }
};
```

- **Concrete Creators:**

```
class ConcreteCreatorA : public Creator {
public:
    std::shared_ptr<Product> FactoryMethod() override {
        return std::make_shared<ConcreteProductA>();
    }
};
class ConcreteCreatorB : public Creator {
public:
    std::shared_ptr<Product> FactoryMethod() override {
        return std::make_shared<ConcreteProductB>();
    }
};
```

4. GoF Abstract Factory

- **Intent:** Definer en grænseflade til at skabe familier af relaterede eller afhængige objekter uden at specificere deres konkrete klasser.
- **Struktur:**
  - **Abstract Factory:**

```
class AbstractFactory {
public:
    virtual std::shared_ptr<ProductA> CreateProductA() = 0;
    virtual std::shared_ptr<ProductB> CreateProductB() = 0;
};
```

  - **Concrete Factories:**

```
class ConcreteFactory1 : public AbstractFactory {
public:
    std::shared_ptr<ProductA> CreateProductA() override {
        return std::make_shared<ConcreteProductA1>();
    }
    std::shared_ptr<ProductB> CreateProductB() override {
        return std::make_shared<ConcreteProductB1>();
    }
};
class ConcreteFactory2 : public AbstractFactory {
public:
    std::shared_ptr<ProductA> CreateProductA() override {
        return std::make_shared<ConcreteProductA2>();
    }
    std::shared_ptr<ProductB> CreateProductB() override {
        return std::make_shared<ConcreteProductB2>();
    }
```

```
        };
```

Disse noter giver en grundlæggende forståelse af designmønstrene Template Method, Strategy, Factory Method og Abstract Factory, samt deres strukturer og anvendelser i C++.