



# DEPENDENCY MANAGEMENT

## INGEGNERIA DEL SOFTWARE

Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica

# DEPENDENCY

---

*The quality or state of being influenced or determined by or subject to another.*

- **Changes** in a component may influence its dependencies
  - Internal changes: implementation
  - External changes: interface or extrinsic behaviour
- Dependency a measure of the **probability** of **changes** among dependent components
  - The stronger the dependency the higher the probability

# COUPLING

---

- A **measure** of the degree of **dependency**
  - Tightly-coupled: higher probability of changes
  - Loosely-coupled: lower probability of changes

*Dependency between components must be minimized, making components loosely coupled.*

*Gang of Four*

- Free to change a component, without introducing bugs
  - Internal / external changes
  - Architecture are dynamic and evolve during time

# DEPENDENCY IN OOP

- Dependencies among types
  - Concrete and abstract classes, interfaces

Name	Description
<i>Dependency</i>	When objects of one class <b>work briefly with</b> objects of another class
<i>Association</i>	When objects of one class <b>work with</b> objects of another class for some prolonged amount of time
<i>Aggregation</i>	When one class <b>owns but shares a reference to</b> objects of another class
<i>Composition</i>	When one class <b>contains</b> objects of another class
<i>Inheritance</i>	When one class <b>is a</b> type of another class

- Lines of code and and time (scope)
  - Let's analyze one by one

# DEPENDENCY (RELATION)

## ○ Weakest form of dependency

- Limited in time: execution of one method
- Limited in shared code: interface only

```
class A {  
    public A() { /* ... */ }  
    public void methodA() { /* ... */ }  
}
```

*Shared code: signature*

```
class B {  
    public void methodWithAParam(A param) {  
        a.methodA();  
    }  
    public A methodThatReturnsA() {  
        return new A();  
    }  
}
```

*Dep. interval*

*Dep. interval*

# ASSOCIATION

- A class contains a reference to an object
  - Spans all over an object life time: **permanent**
    - Impacts also object construction
  - All behaviours of a class are virtually shared

```
class A {  
    private B b;  
    public A(B b) { this.b = b; }  
    // Other methods of class A  
}
```

*Dep. interval*

```
class B {  
    public void method1() { /* ... */ }  
    public void method2() { /* ... */ }  
    public void method3() { /* ... */ }  
}
```

*Every method  
signature of B*

# AGGREGATION AND COMPOSITION

- One type owns the other
  - Addition of creation and deletion responsibility
    - Creation is not a simple affair...
  - Composition: avoid shareability of components

```
class A {  
    private B b;  
    public A() {  
        // A must know how to build a B  
        this.b = new B("param1", "param2");  
    }  
    // Other methods of class A  
  
    static class B {  
        // Rest of the class  
        public B(String param1, String param2) { /* ... */ }  
    }  
}
```

*Dep. interval*

*Also the building process is shared*

# INHERITANCE

- Strongest type of dependency
  - Inheritance and reuse of the not private code
    - (Implementation inheritance, not subtyping)
  - Any change to the parent can disrupt its children

```
class A {  
    public A() { /* ... */ }  
    // Other methods of class A  
}
```

*Shared code*

```
class B extends A {  
    public B() {  
        super();  
        // ...  
    }  
    // Other methods of class B  
}
```

*Dep. interval*



# DEPENDENCY DEGREE

- The more the **shared code**, the stronger the dependency
  - Also, the wider the scope, ...
- Can we formalize a **measure of coupling**,  $\delta_{A \rightarrow B}$ ?

$$\delta_{A \rightarrow B} = \frac{\varphi_{S_{A|B}}}{\varphi_{S_{tot_B}}} \varepsilon_{A \rightarrow B} \in \{x \in \mathbb{R}^+ | 0 \leq x \leq 1\}$$

- $\varphi_{S_{A|B}}$ : SLOC shared between A and B
- $\varphi_{S_{tot_B}}$ : Total SLOC of class B
- $\varepsilon_{A \rightarrow B}$ : A factor  $[0, 1]$  that measures the scope

# DEPENDENCY DEGREE

---

- Coupling is proportional to the **probability** of mutual **change** between components

$$\delta_{A \rightarrow B} \propto P(B_{mod} | A_{mod})$$

- Measure of total coupling for a component

$$\delta^A_{tot} = \frac{1}{n} \sum_{C_j \in C_1, \dots, C_n} \delta_{A \rightarrow C_j}$$

- $C_j$  is the  $j$ th class A depends on
- The measure is the **mean** of all coupling measures

# INFORMATION HIDING

---

- Remember the `Rectangle` class?
  - What if height and width have their own types?
    - Height, Width and `Rectangle` types would always be used together
  - They are **tightly-coupled**
  - The  $\delta^C_{tot}$  of a client C would be very high
    - It would use always all the three types
  - $\delta^{Rectangle}_{tot}$  would be high too
- The given solution probably obtains the **minimization** of  $\delta^C_{tot}$

# REFERENCES

---

- Dependency.

<http://rcardin.github.io/programming/oop/software-engineering/2017/04/10/dependency-dot.html>

- The Secret Life of Objects: Information Hiding

<http://rcardin.github.io/design/programming/oop/fp/2018/06/13/the-secret-life-of-objects.html>

# GITHUB REPOSITORY

---



<https://github.com/rcardin/swe>