Lecture: SOLID Principles & Inter-class Relationships

ESS201: Programming II

Jaya Sreevalsan Nair, IIIT-Bangalore

August 19, 2019

[Recap] Features of OOP (Object-Oriented Programming)

Summary: Features of OOP

Encapsulation: Reduce complexity + increase reusability

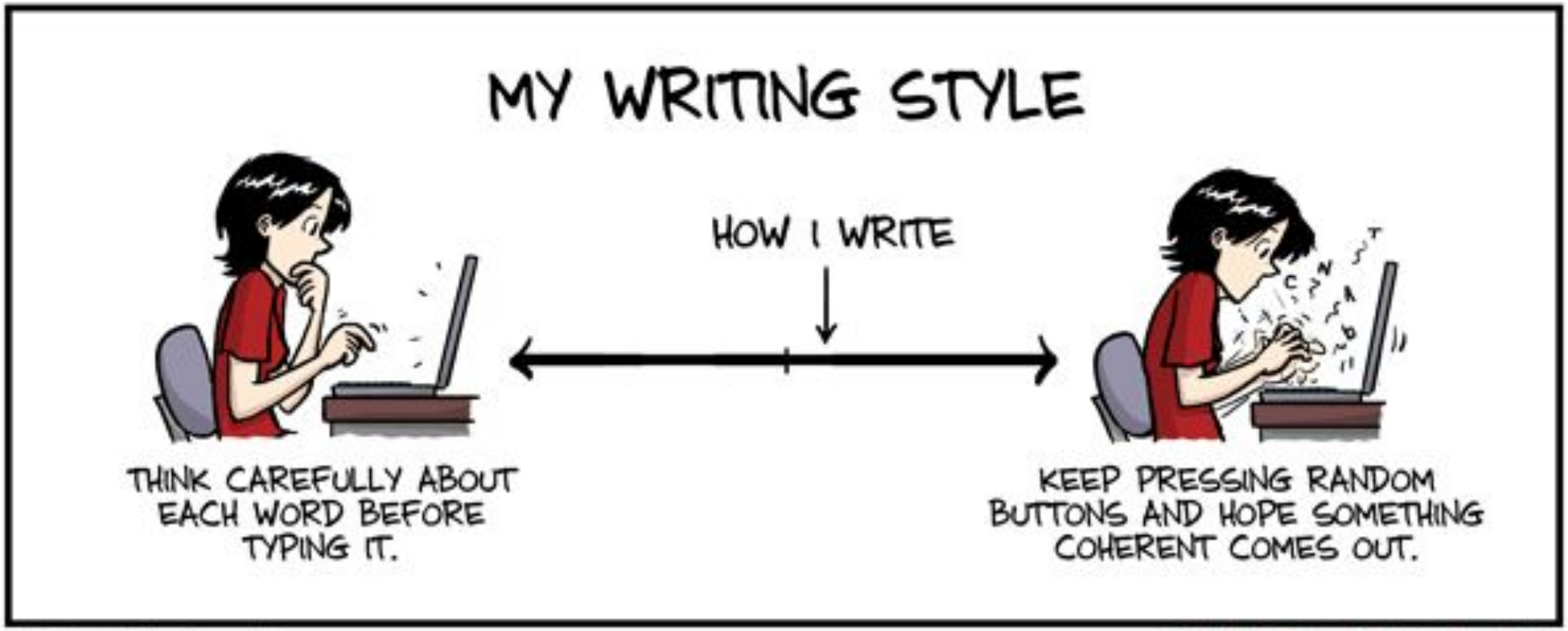
Abstraction: Reduce complexity + isolate impact of changes

Inheritance: Eliminate redundancy

Polymorphism: Refactor unnecessary inflexible switch/case statements

Courtesy:https://www.youtube.com/watch?v=pTB0EiLXUC8

SOLID Principles



http://phdcomics.com/comics/archive.php?comicid=1919

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

Introduced by Robert Martin (author of “Clean Code”).

Originators of LSP is Barbara Liskov; and OCP is Bertrand Meyer.

Read more at: https://stackify.com/solid-design-principles/

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)

  "A class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class."

- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)

  "Increase the cohesion between things that change for the same reasons, decrease the coupling between those things that change for different reasons."

- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)

"Software entities ... should be open for extension, but closed for modification."

- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)

"A class is closed, since it may be compiled, stored in a library, baselined, and used by client classes. But it is also open, since any new class may use it as parent, adding new features. When a descendant class is defined, there is no need to change the original or to disturb its clients."

- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)

"Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program."

- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)

"It extends the Open/Closed principle and enables you to replace objects of a parent class with objects of a subclass without breaking the application. "

- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)

  “Many client-specific interfaces are better than one general-purpose interface."

- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)

  “Clients should not be forced to depend upon interfaces that they do not use."

- **D** stands for **DIP** (Dependency inversion principle)

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

One should "depend upon abstractions, [not] concretions." Closely related to Hollywood Principle, “Don’t call us. We’ll call you.”

SOLID Principles

SOLID is an acronym which stands for the five basic principles which help to create good software architecture.

- **S** stands for **SRP** (Single responsibility principle)
- **O** stands for **OCP** (Open closed principle)
- **L** stands for **LSP** (Liskov substitution principle)
- **I** stand for **ISP** (Interface segregation principle)
- **D** stands for **DIP** (Dependency inversion principle)

“High-level modules should not depend on low-level modules. Both should depend on abstractions.”

“Abstractions should not depend on details. Details should depend on abstractions.”

Inter-class Relationships

Introduction

Once we have created classes in OOP, we can now have these classes interact in a given context. We will refer to these interactions as "inter-class relationships."

Types of inter-class relationships:

1. Composition
2. Aggregation
3. Association

Introduction

Once we have created classes in OOP, we can now have these classes interact in a given context. We will refer to these interactions as "inter-class relationships."

Types of inter-class relationships:

1. Composition: the object and its member have a strong part-whole relationship; characterized by "part-of" relationship.
2. Aggregation: a weaker form of composition, where the object and its member form a "has-a" relationship.
3. Association: a weaker form of aggregation, where the object and its member form a "uses-a" relationship.

Introduction

Once we have created classes in OOP, we can now have these classes interact in a given context. We will refer to these interactions as "inter-class relationships."

Types of inter-class relationships:

1. Composition: The motherboard is a "part-of" the computer.
2. Aggregation: The desktop "has-a" monitor .
3. Association: The student "uses-a" supercomputer.

Property	Composition	Aggregation	Association
Relationship type	Whole/part	Whole/part	Otherwise unrelated
Members can belong to multiple classes	No	Yes	Yes
Members existence managed by class	Yes	No	No
Directionality	Unidirectional	Unidirectional	Unidirectional or bidirectional
Relationship verb	Part-of	Has-a	Uses-a

Image courtesy: https://www.learncpp.com/cpp-tutorial/10-4-association/

Composition vs Aggregation

In composition, part does not survive without the whole; unlike aggregation.

```
// composition
class Whole {
   Part part;
   // ...
};
```

```
// shared aggregation
class Whole {
   Part* part;
   // ...
};
```

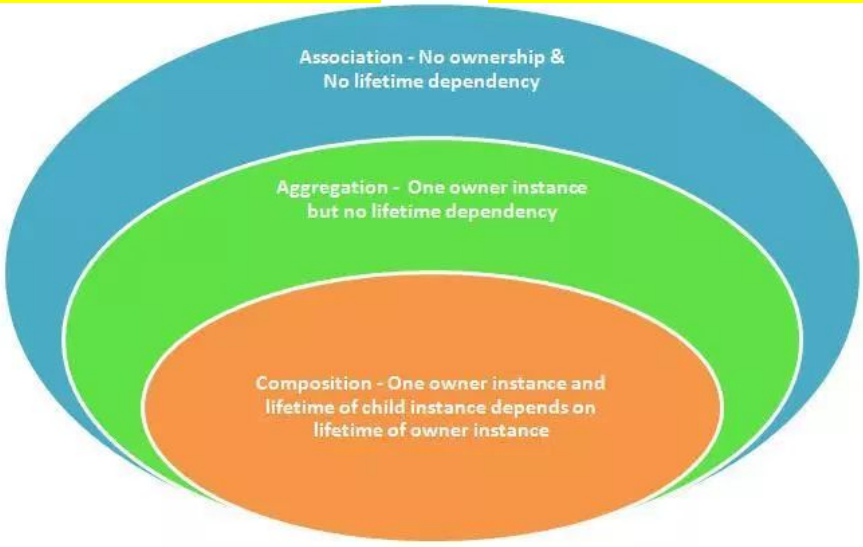


Image courtesy: https://dotnetfreakblog.wordpress.com/2014/01/11/concept-of-dependency-generalization-association-aggregation-composition-in-object-oriented-programming/