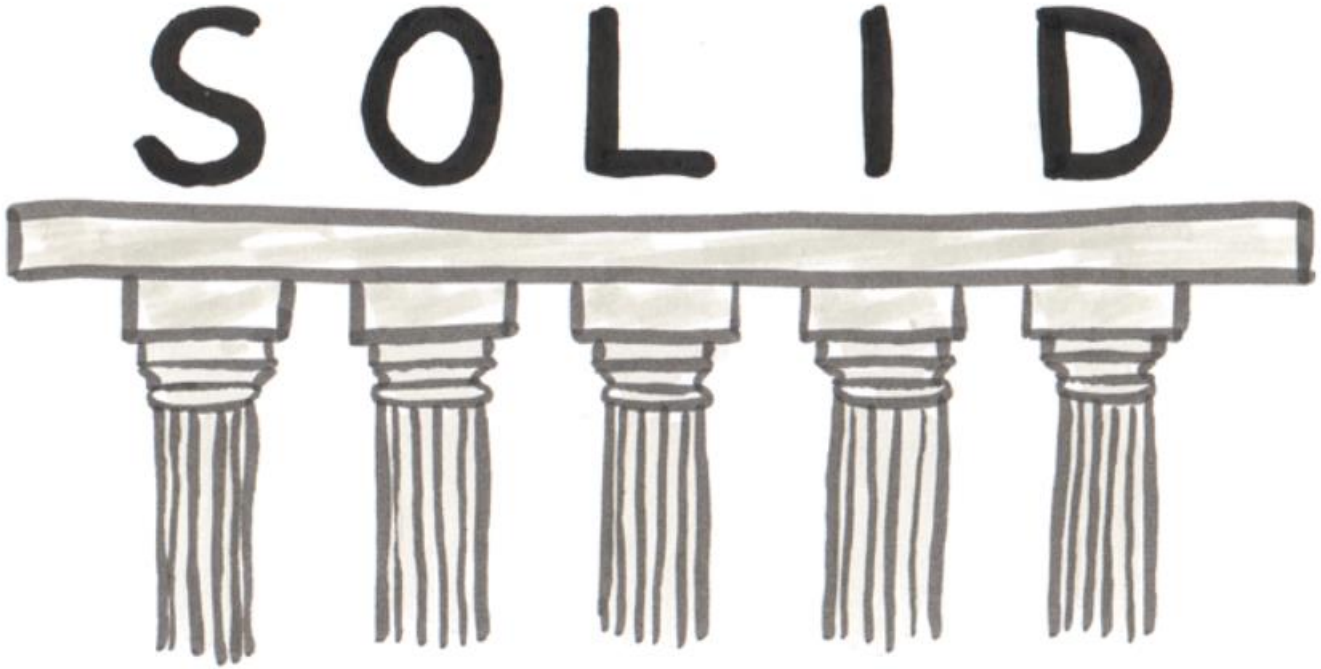
PROGRAMAÇÃO EM C++
PROJETO FINAL

INF1900

Prof. Dr. Bruno B. P. Cafeo

Institute of Computing
University of Campinas

Agenda



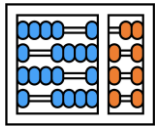
Single Responsibility Principle

- Cada módulo ou classe deve ter responsabilidade sobre uma única parte da funcionalidade fornecida pelo software.

- Essa responsabilidade deve ser totalmente encapsulada pela classe.

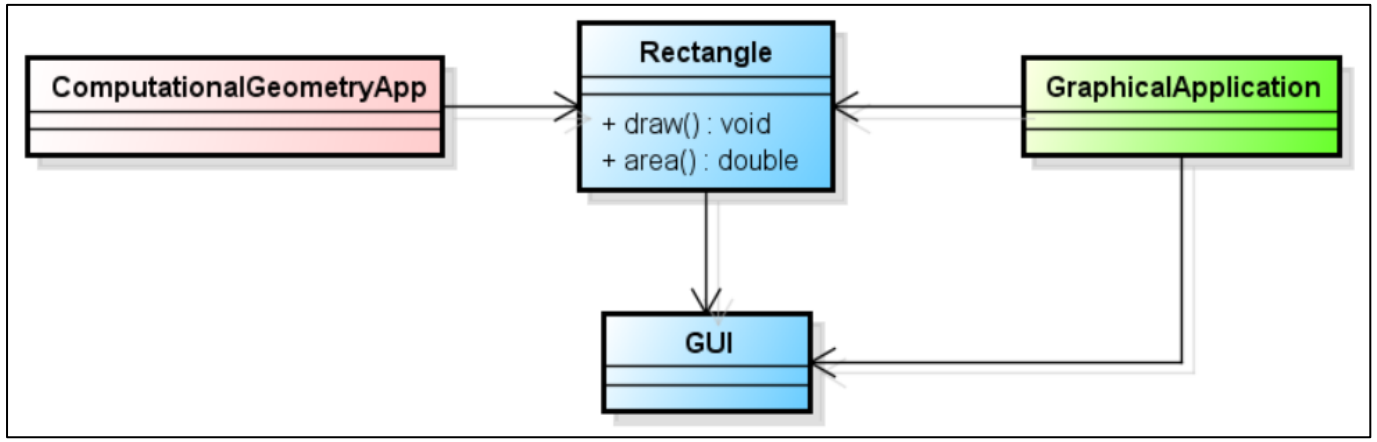
Princípios de SRP podem ser usados quando

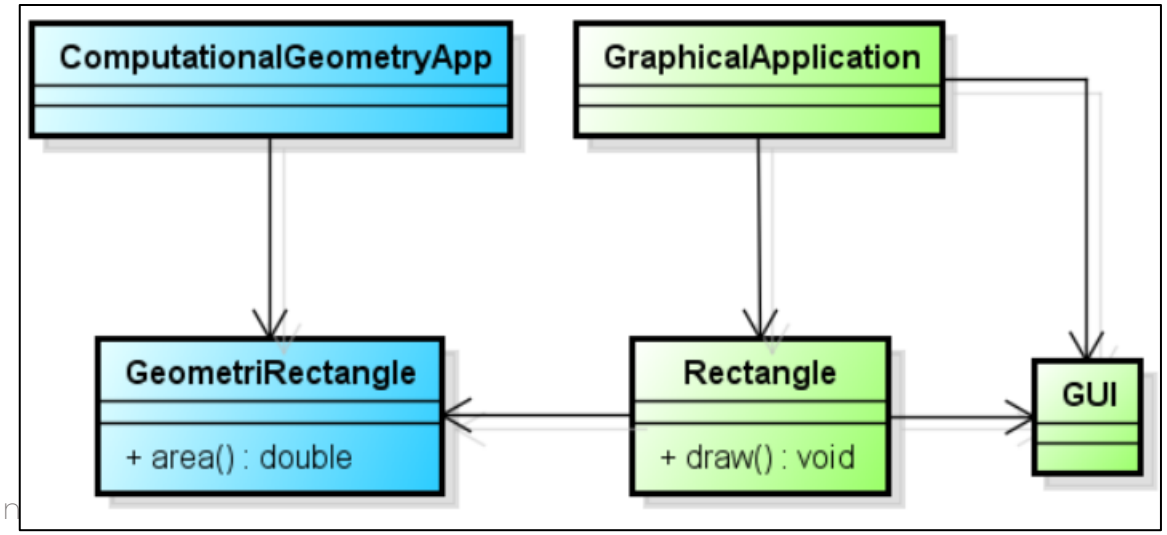
- Muita coisa é permitida ao objeto da classe.
- Qualquer mudança na lógica do comportamento do objeto resulta em alterações em outros locais da aplicação.
- Você precisa testar, corrigir erros, mesmo que terceiros sejam responsáveis pelo desempenho deles.
- Não é tão simples separar e aplicar uma classe em outra área da aplicação, já que isso trará dependências desnecessárias.



UNICAMP

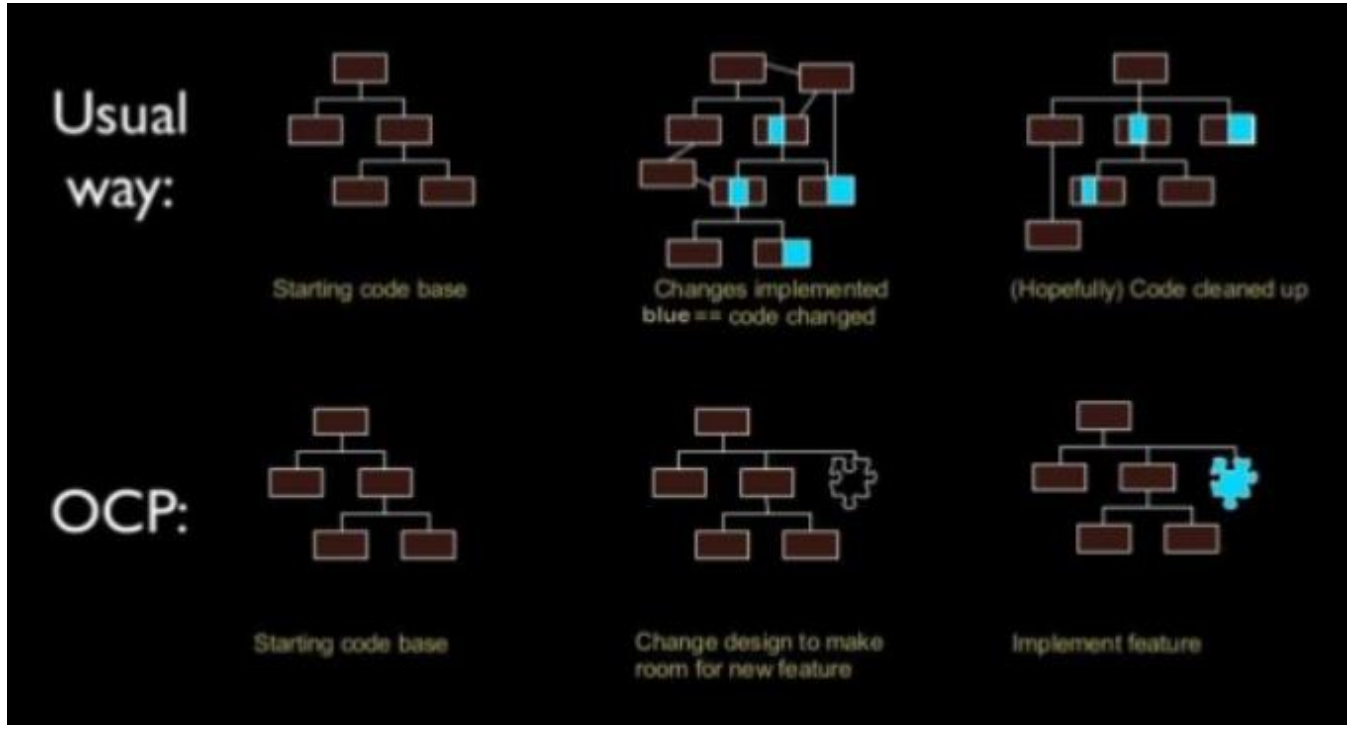
Ruim vs. Bom


ComputationalGeometryApp
Rectangle
+ draw() : void
+ area() : double
GraphicalApplication
GUI


ComputationalGeometryApp
GraphicalApplication
GeometriRectangle
+ area() : double
Rectangle
+ draw() : void
GUI

Open-Closed Principle

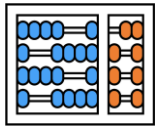
- Entidades de software (classes, módulos, funções, etc.) devem estar abertas para extensão, mas fechadas para modificação.



OCP sugere que entidades devem ser

- Abertas para extensão: isso significa que o módulo pode ser expandido. Quando os requisitos da aplicação mudam, somos capazes de ampliar o módulo.
  - Em outras palavras, temos a capacidade de estender classes, tornando-as mais funcionais. Ao mesmo tempo, o comportamento dos métodos antigos não muda, e a classe em si não sofre alterações.

- Fechadas para modificação: após a expansão do comportamento da entidade, não devem ser feitas alterações no código que utiliza essas entidades.

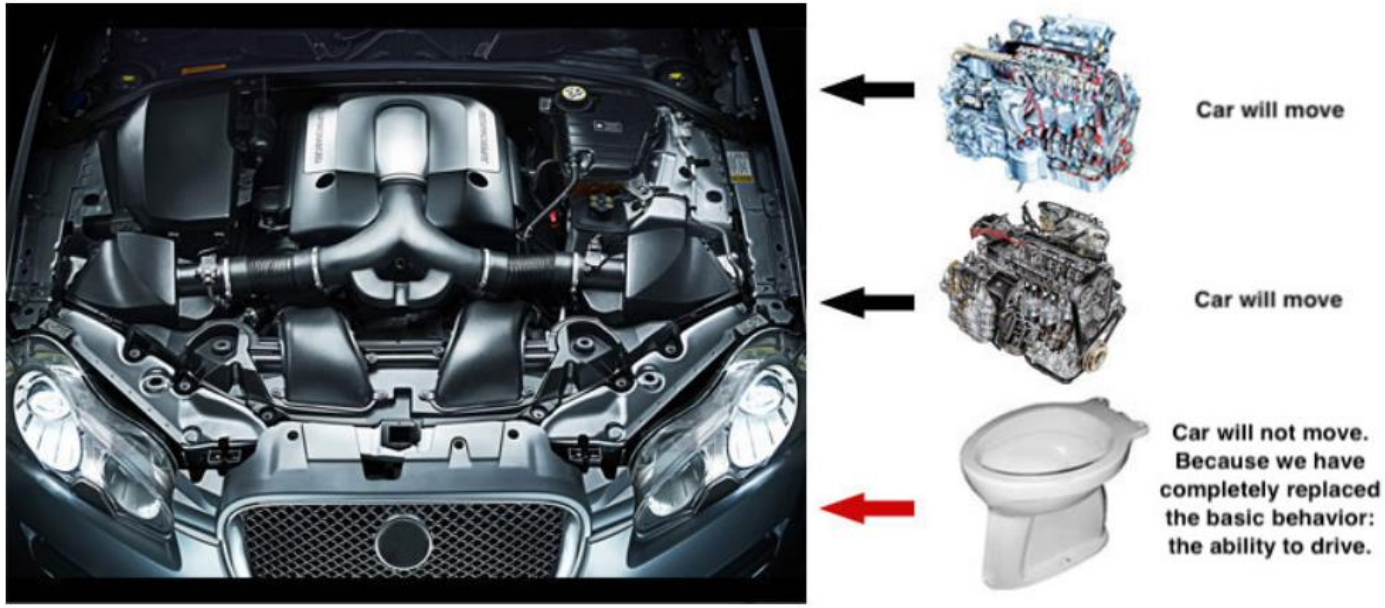


UNICAMP

Liskov Substitution Principle

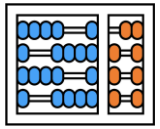
- Liskov's notion of a behavioural subtype defines a notion of substitutability for objects.

- Se S for um subtipo de T, então objetos do tipo T em um programa podem ser substituídos por objetos do tipo S sem alterar quaisquer propriedades desejáveis desse programa (por exemplo, corretude).



LSP é seguido nos seguintes casos

- Nenhuma nova exceção deve ser lançada pelos métodos do subtipo, exceto quando essas exceções são elas próprias subtipos das exceções lançadas pelos métodos do supertipo.
- Não viola a funcionalidade.
- Retorna o mesmo tipo.
- O objeto da subclasse possui um contrato com a superclasse.



UNICAMP

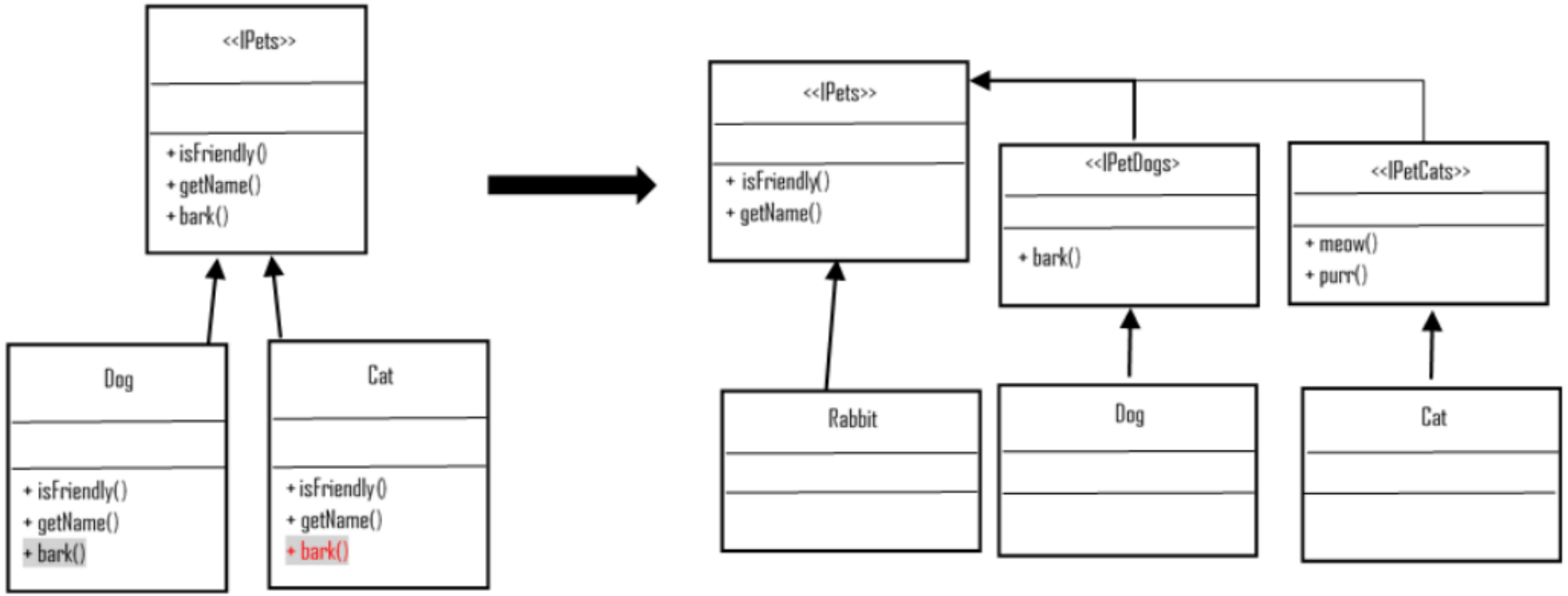
Interface Segregation Principle

- Nenhum cliente deve ser obrigado a depender de métodos que não utiliza.

- Nenhum cliente deve ser obrigado a depender de métodos que não utiliza.

- O ISP tem a intenção de manter um sistema desacoplado, tornando-o assim mais fácil de refatorar, alterar e redistribuir.

Exemplo

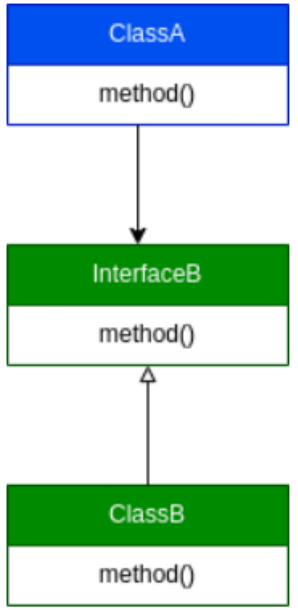

<<IPets>>
+ isFriendly()
+ getName()
+ bark()
Dog
+ isFriendly()
+ getName()
+ bark()
Cat
+ isFriendly()
+ getName()
+ bark()
<<IPets>>
+ isFriendly()
+ getName()
<<IPetDogs>>
+ bark()
<<IPetCats>>
+ meow()
+ purr()
Rabbit
Dog
Cat

Dependency Inversion Principle

- Módulos de alto nível não devem importar nada de módulos de baixo nível. Ambos devem depender de abstrações (por exemplo, interfaces).

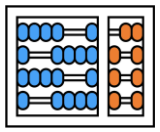
- Abstrações não devem depender de detalhes. Detalhes (implementações concretas) devem depender de abstrações.

Exemplo

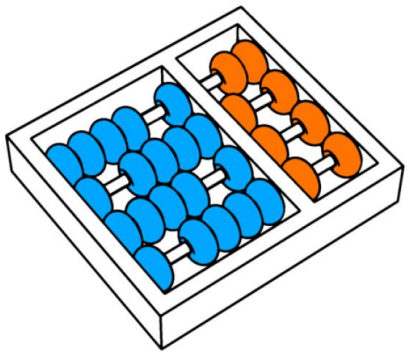



```
class ClassB {
    // fields, constructor and methods
}

class ClassA {
    ClassB objectB;

    ClassA(ClassB objectB) {
        this.objectB = objectB;
    }
    // invoke classB methods
}
```

```
interface InterfaceB {
    method()
}

class ClassB implements InterfaceB {
    // fields, constructor and methods
}

class ObjectA {
    InterfaceB objectB;

    ObjectA(InterfaceB objectB) {
        this.objectB = objectB;
    }
    ...
}
```



UNICAMP



INSTITUTO DE COMPUTAÇÃO

UNICAMP

Prof. Dr. Bruno B. P. Cafeo

Sala 04
Instituto de Computação - Unicamp
Av. Albert Einstein, 1251
Cidade Universitária
Campinas – SP
13083-852

https://ic.unicamp.br/~cafeo/
cafeo@ic.unicamp.br