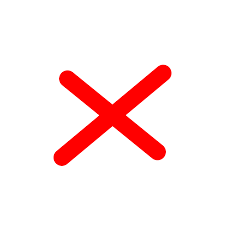
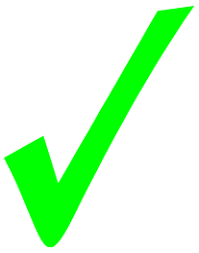
**SOLID**

A programação baseada nos princípios SOLID promovem uma boa forma de construção de código, pois visam separar as responsabilidades e a diminuição de acoplamento entre as classes. É uma boa prática na Programação Orientada a Objetos.

Robert Martin, em um artigo, elaborou cinco técnicas de programação orientada a objetos onde cada técnica é uma das letras da palavra SOLID. Esses cinco princípios são:

* ***Single Responsibility Principle***: princípio da responsabilidade única;



Uma classe deve ter apenas um objetivo, deve possuir uma função ou funções para resolver somente uma tarefa, mantendo uma única responsabilidade. A quebra desta responsabilidade pode significar diversos problemas de implementação. Após a primeira construção do código, se houver a necessidade de alguma alteração no projeto do software, possivelmente as classes teriam que ser refeitas, ou então o uso de condições seria utilizado para distinguir o ambiente em si.

* ***Open Closed Principle***: princípio do aberto/fechado;

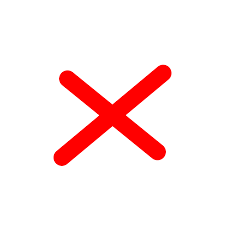
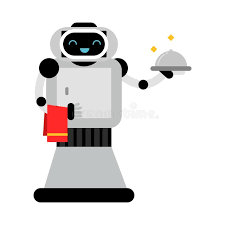
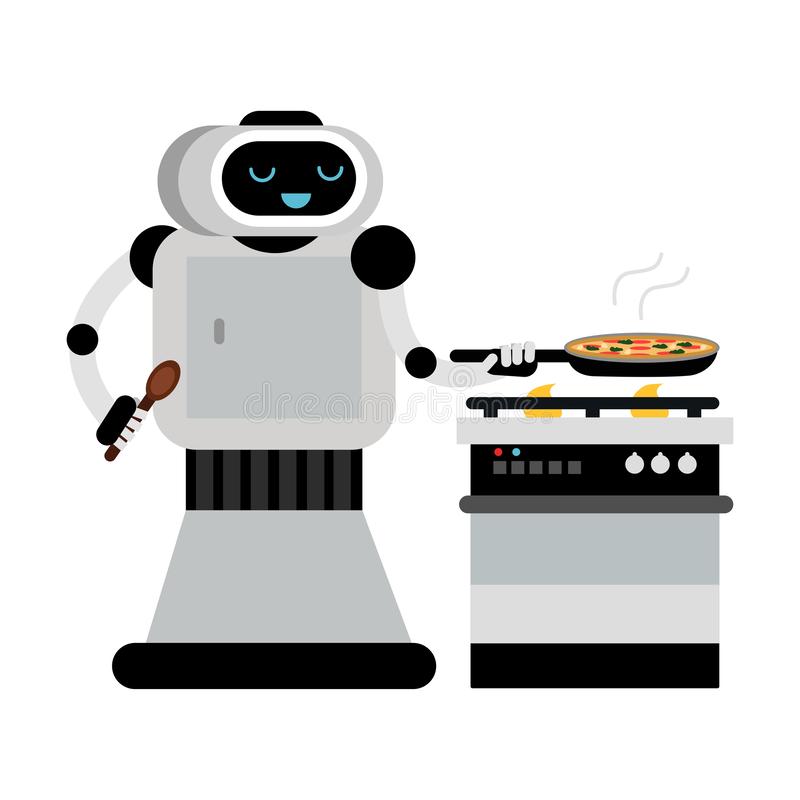
Este princípio traz a ideia de que as classes da aplicação devem ser abertas para extensões e fechadas para modificações.

Alterar uma classe pai poder gerar problemas porque outras classes podem estar utilizando a mesma.

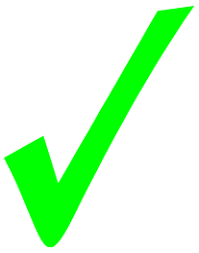
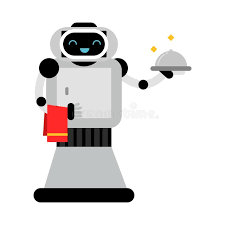
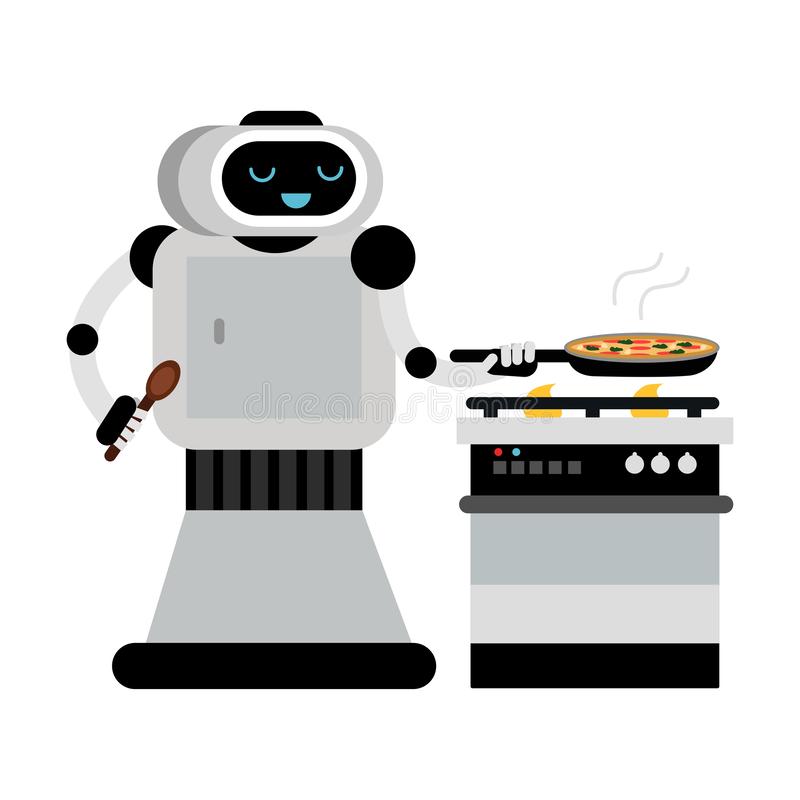
Na prática, quando realizamos alguma alteração de recurso ou de comportamento da entidade, devemos estender este novo recurso e comportamento e não fazer alterações no código-fonte.

Não cozinho mais, agora eu só sirvo

Eu cozinho



Eu já cozinhava. Agora posso cozinhar e servir



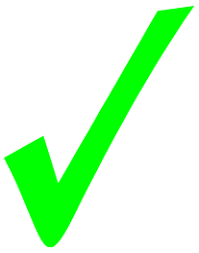
* ***Liskov Substitution Principle***: princípio da substituição de Liskov;

Barbara Liskov, cientista americana, trouxe a ideia de que as classes derivadas devem ser substituíveis pelas suas classes bases.

Este princípio trabalha com a ideia de herança, onde a classe pai tem os seus atributos genéricos, e uma classe filha herda estes atributos e poder ter outros específicos próprios.

Devemos ficar atentos para não sobrescrever ou implementar um método que não faz nada, lançar uma exceção e retornar tipos e valores diferentes da classe pai.

Eu tenho RG e CPF

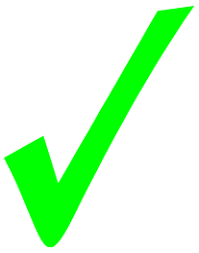


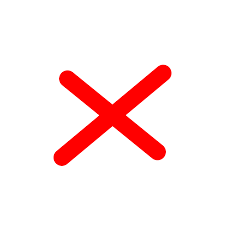
Eu tenho RG, CPF e carteirinha de estudante

* ***Interface Segregation Principle****:*princípio da segregação de *Interfaces*;

Neste princípio de segregação de interface, é preciso**implementar somente o que importa para as nossas classes,** pois não se deve forçar uma classe a implementar métodos que não são úteis para ela, isso não é uma boa prática de construção de código.

Então o ideal é quebrar uma interface em outras menores e específicas ao invés de interfaces genéricas.

 cada uma com a sua função

 uma para resolver tudo 

* ***Dependency Inversion Principle****:*princípio da inversão de dependência.

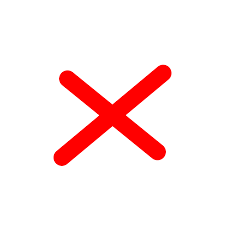
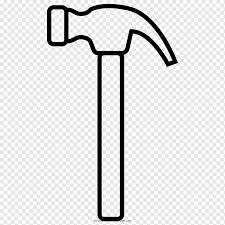
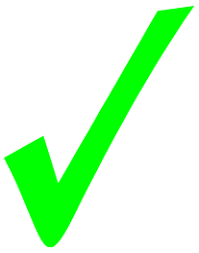
Os módulos que são classes de alto nível devem depender de conceitos, e não de módulos de baixo nível, ambos devem depender das abstrações.

**E as abstrações não devem depender de detalhes. Os detalhes devem depender das abstrações.**

O que estamos falando aqui, é do princípio do baixo acoplamento, onde a dependência entre as classes da composição é baixa. Conseguimos isso através de abstrações ao invés das classes concretas.

Corto a pizza com o cortador de pizza

Corto pizza com a ferramenta que tiver



Em POO, utilizar o SOLID deixa o código mais limpo, pois promove a segurança, a manutenção e a reutilização do código.

Utilizar os princípios SOLID torna o software mais robusto, escalável, flexível e tolerante a mudanças, pois facilita a implementação de novos requisitos para evolução e manutenção do sistema.