Aula JavaScript

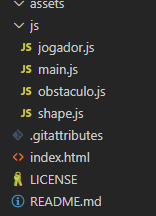
Os Princípios SOLID de Design

O termo ***SOLID***é um ***acrônimo*** para cinco postulados de design, destinados a facilitar a ***compreensão***, o ***desenvolvimento*** e a ***manutenção*** de software. Eles são cinco:

**Princípio de única responsabilidade**

*"Uma [classe](https://pt.wikipedia.org/wiki/Classe_(programa%C3%A7%C3%A3o)" \o "Classe (programação)) deve ter apenas uma única responsabilidade (mudanças em apenas uma parte da especificação do software, devem ser capaz de afetar a especificação da classe)."*

Em outras palavras, cada ***classe*** deve ser ***responsável*** por apenas uma coisa, e ele deve ser “***destacado***” de nosso ***código principal***, apenas sendo chamada. Então, por exemplo, se tivéssemos uma classe responsável por uma forma geométrica, o código dela não estaria junto com o principal, mas em um ***arquivo separado***, sendo apenas importado, isso deixa o código mais fácil de ler e realizar manutenção.

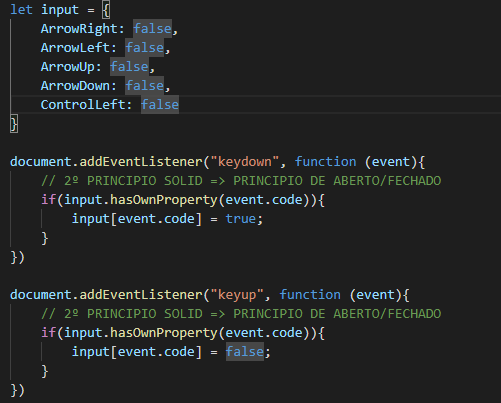


Cada arquivo representa uma classe e possui uma única responsabilidade.

**Princípio de aberto/fechado**

*"Entidades de software devem ser abertas para extensão, mas fechadas para modificação."*

Isso basicamente nos diz que nosso código deve ser ***extremamente genérico***, onde para adicionarmos algo precisamos apenas dizer sua funcionalidade e ele irá funcionar.

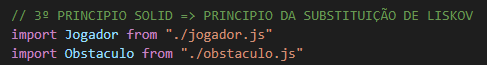


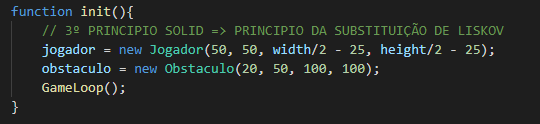
Em nosso código precisamos apenas adicionar uma nova tecla na lista de input e na lógica do jogo dizer o que essa tecla faz, pronto :D

**Princípio da substituição de Liskov**

*"Objetos em um programa devem ser substituíveis por instâncias de seus subtipos, sem alterar a funcionalidade do programa. Deve ser capaz de afetar apenas a especificação da classe"*

Em outras palavras, se tivermos dois, ou mais, objetos que usam as mesma classe, mas tem ***funcionalidades diferentes***, devemos ***criar*** mais duas classes que são ***subtipos/filhos*** da anterior responsáveis apenas por esses dois novos objetos.  
 Em nosso jogo, por exemplo, temos o jogador e o obstáculo, ambos fazem parte da classe *shape*, entretanto, possuem métodos e funcionalidades diferentes. Por isso devemos separa-los em duas classes diferentes.





Os dois últimos princípios: **Princípio da segregação de Interface** e **Princípio da inversão de dependência** serão vistos com mais calma durante o curso.