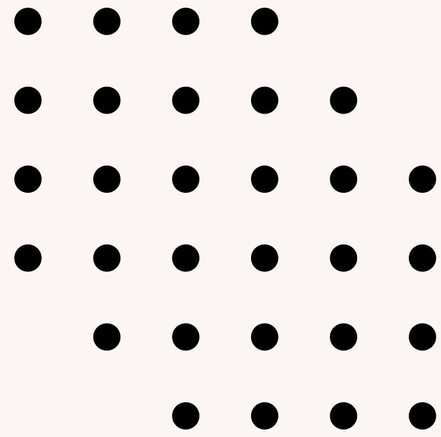
DESIGN DE CÓDIGO

Isabelle Brilhante

SOLID

Cinco princípios de design de código da programação orientada a objetos

- **S**ingle Responsability
- **O**pen Closed
- **L**iskov Substitution
- **I**nterface Segregation
- **D**ependency Inversion



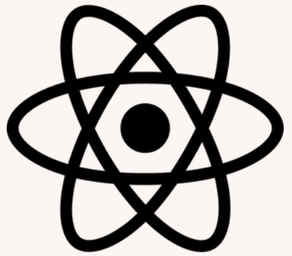
SINGLE RESPONSABILITY (RESPONSABILIDADE ÚNICA)

“Uma classe deve ter um, e somente um, motivo para mudar.”

- Classes devem ter apenas uma responsabilidade
- Reduz complexidade e facilita manutenção.
- Evita classes "faz-tudo"
- Caso esse princípio não seja seguido:
  - Alto acoplamento
  - Dificuldade na criação de testes
  - Pode aumentar a duplicação de código

RESPONSABILIDADE ÚNICA EM REACT

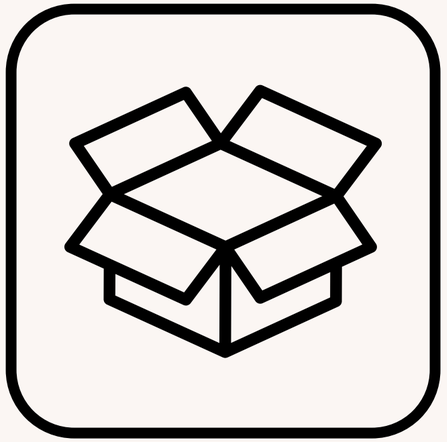
- Dividir componentes grandes em componentes menores.
- Isolar código não relacionado em funções de utilitário separadas.
- Agrupar funcionalidades relacionadas em hooks personalizados.
- Componentes de Apresentação e Container
  - Apresentação - Trata da aparência das coisas
  - Container - Trata como as coisas funcionam



OPEN CLOSED (ABERTO–FECHADO)

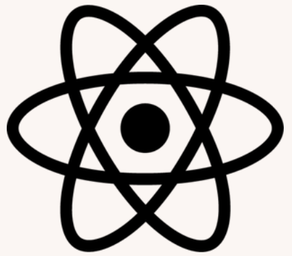
“Objetos ou entidades devem estar abertos para extensão, mas fechados para modificação.”

- Extensão: Novos métodos.
- Modificação: Alterações em métodos existentes.
- Criar extensões para introduzir novos comportamentos.
- Modificar comportamentos já existentes pode causar bugs.



ABERTO-FECHADO EM REACT

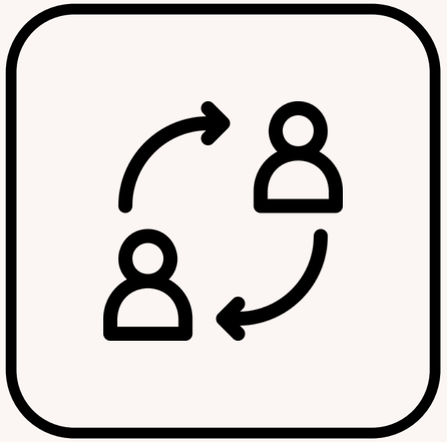
- Fazer alterações sem que mude a forma como o componente é utilizado
- Útil para componente comum usado em muitos lugares.
- Reduz acoplamento entre componentes.
- Aumenta extensibilidade e reutilização.



LISKOV SUBSTITUTION (SUBSTITUIÇÃO DE LISKOV)

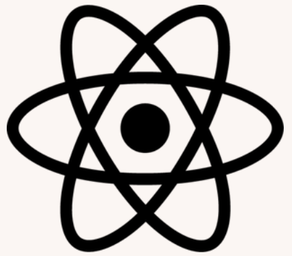
“Uma classe derivada deve ser substituível por sua classe base.”

- O comportamento deve ser igual ao trocar um pai por um filho
- Garante consistência e previsibilidade na substituição de objetos de diferentes classes.



SUBSTITUIÇÃO DE LISKOV EM REACT

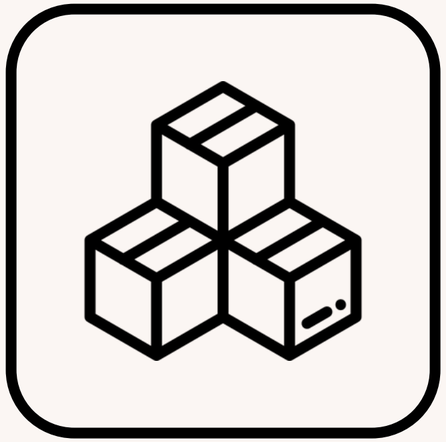
- Poder substituir um componente pelo seu subtipo
- Útil em casos que o componente deve substituir o componente base ou elemento que estende
- Caso contrário, deve ser evitado



INTERFACE SEGREGATION (SEGREGAÇÃO DA INTERFACE)

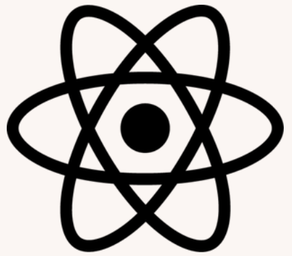
“Uma classe não deve ser forçada a implementar interfaces e métodos que não irá utilizar.”

- Composições de interfaces podem simplificar a implementação
- Não criar uma única interface grande.
- Preferir interfaces menores e focadas em casos específicos.
- Pode melhorar a organização e reusabilidade do código.



SEGREGAÇÃO DA INTERFACE EM REACT

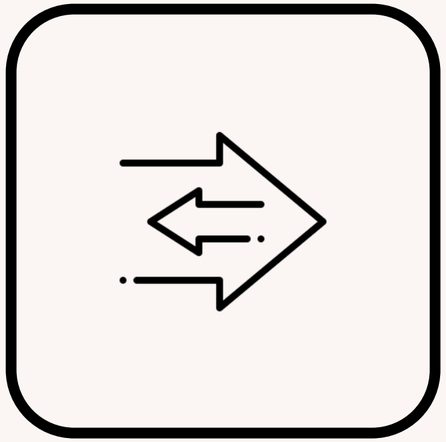
- Componentes não devem depender de props não utilizadas.
  - Ex: Uso de um tipo que traga propriedades não utilizadas
    - Depende de detalhes não necessários
- Reduz as dependências entre os componentes.
- Diminui o acoplamento, promovendo a reutilização.



DEPENDENCY INVERSION (INVERSÃO DE DEPENDÊNCIA)

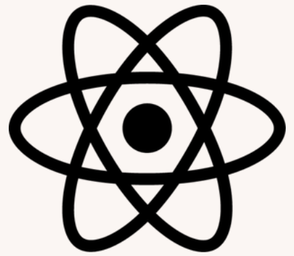
“Dependa de abstrações e não de implementações.”

- Módulos de alto nível não devem importar módulos de baixo nível. Ambos devem depender de abstrações (interfaces).
- Abstrações não devem depender de detalhes. Detalhes (implementações concretas) devem depender de abstrações.
- Permite substituir objetos.
- Evita acoplamento excessivo.
- Facilita manutenção, testes e extensões de código.



INVERSÃO DE DEPENDÊNCIA EM REACT

- Dependência - código externo usado no componente
- Importação ou injeção de parâmetro/prop
- O Padrão de Composição em React e o Princípio de Inversão de Dependência são equivalentes
- No contexto de React, a inversão de dependência envolve transferir responsabilidade para o componente pai.



ARTIGOS

SOLID

- https://www.freecodecamp.org/news/solid-principles-for-better-software-design/
- https://medium.com/desenvolvendo-com-paixao/o-que-%C3%A9-solid-o-guia-completo-para-voc%C3%AA-entender-os-5-princ%C3%ADpios-da-poo-2b937b3fc530

ARTIGOS

SOLID EM REACT

- https://itnext.io/solid-in-react-the-good-the-bad-and-the-awesome-79d6cc518d1f
- https://javascript.plainenglish.io/solid-principle-in-react-11272c41b529
- https://www.codigofonte.com.br/artigos/aplicando-principios-solid-em-react

Obrigada por assistir
:D