🧠 1. Principles of Clean Code (Core Mindset)

✅ DRY — *Don't Repeat Yourself*

- **What juniors do:** Copy-paste similar code blocks.
- **What seniors do:** Abstract common logic into functions/components.
- **React example:**

tsx
CopyEdit

```
// ✗ Repetition
<button onClick={() => buyProduct(1)}>Buy</button>
<button onClick={() => buyProduct(2)}>Buy</button>

// ✓ DRY
const BuyButton = ({ id }: { id: number }) => (
  <button onClick={() => buyProduct(id)}>Buy</button>
);
```

✅ KISS — *Keep It Simple, Stupid*

- **What juniors do:** Overcomplicate logic or premature abstraction.
- **What seniors do:** Write straightforward, readable code.
- **React example:**

tsx
CopyEdit

```
// ✗ Complex and nested
{user ? (user.role === 'admin' ? <AdminPanel /> : <UserPanel />) :
null}

// ✓ Simpler and more readable
if (!user) return null;
```

```
return user.role === 'admin' ? <AdminPanel /> : <UserPanel />;
```

✅ SOLID (Object-Oriented principles applied in functions/components)

Principle	Meaning	Example in React
Single Responsibility	One reason to change	Split large components
Open/Closed	Open for extension, closed for modification	Use composition, not modification
Liskov Substitution	Subtypes must be substitutable	Keep props API consistent
Interface Segregation	Don't force unnecessary props	Separate small components
Dependency Inversion	Depend on abstractions	Use hooks/contexts instead of tightly coupling

---

🛠️ 2. Refactoring Techniques

Technique	Description	Before	After
Function Extraction	Break long functions into smaller ones	`handleSubmit()` does 10 things	Split into `validate()`, `sendData()`, `clearForm()`
Component Decomposition	Split large components	`Dashboard.tsx` is 300+ lines	Split into `Header`, `Stats`, `RecentOrders`
Use Hooks	Abstract logic from UI	Logic and rendering mixed	Move logic to custom hook
UseMemo/UseCallback	Prevent re-renders	Expensive function in render	Memoize it
Constants File	Avoid magic strings	`"admin"` everywhere	`ROLES.ADMIN` constant

Utils/Helpers	Reusable logic	Validation logic in form	Move to `validators.ts`

🧹 3. Clean React/Next.js Component Checklist

✅ What to do inside your functional components:

- Keep them **pure**: props in → JSX out.
- Use hooks at the top: `useState`, `useEffect`, etc.
- Extract logic to custom hooks when it gets complex.
- Use early returns instead of nested conditionals.
- Name components and props descriptively.
- Keep JSX structure flat and readable (no deep nesting).

❌ What to avoid inside components:

- Business logic mixed with rendering.
- Side effects inside JSX (e.g. calling functions directly in render).
- Deeply nested ternaries.
- Bloated useEffects (split or encapsulate).
- Hard-coded values and strings.
- Anonymous functions in `onClick` (if performance matters).

🧯 4. Mistakes Junior Developers Make

Mistake	Why it's a problem	How to fix

Mixing logic and UI	Hard to test and reuse	Separate with hooks/helpers
Not breaking down components	Leads to spaghetti code	Use atomic design: small, medium, large components
Overusing state	Adds complexity	Use derived data instead
Naming badly (`x`, `temp`, `data1`)	Reduces readability	Be descriptive and consistent
No folder structure	Hard to scale	Use domain-based or atomic folders
Ignoring performance	Re-renders and slow apps	Learn `useMemo`, `React.memo`, lazy loading

---

📦 5. Example Refactor

Before:

tsx
CopyEdit

```
const Cart = ({ cartItems }) => {
  return (
    <div>
      {cartItems.map((item) => (
        <div key={item.id}>
          <img src={item.image} />
          <h2>{item.title}</h2>
          <p>{item.price}</p>
          <button onClick={() =>
removeFromCart(item.id)}>Remove</button>
        </div>
      ))}
    </div>
  );
};
```

After:

tsx

CopyEdit

```
// CartItem.tsx
const CartItem = ({ item, onRemove }) => (
  <div className="cart-item">
    <img src={item.image} />
    <h2>{item.title}</h2>
    <p>{item.price}</p>
    <button onClick={() => onRemove(item.id)}>Remove</button>
  </div>
);

// Cart.tsx
const Cart = ({ cartItems }) => (
  <div>
    {cartItems.map((item) => (
      <CartItem key={item.id} item={item} onRemove={removeFromCart} />
    ))}
  </div>
);
```

Benefits:

- Reusable `CartItem` component.
- Cleaner `Cart` code.
- Easier to test and debug.

📘 Recommended Resources

Books:

- **Clean Code** by Robert C. Martin *(must read)*
- **Refactoring** by Martin Fowler *(practical examples)*
- **The Pragmatic Programmer**