Developer Notes – Core Java + Spring + React (Conceptual + Interview Ready)

□ CORE JAVA – OOP + STATIC + CONSTRUCTOR + BLOCKS

□ 1. Class & Object (Basics)

- **Class** = blueprint.
- **Object** = instance of a class.
- Everything in Java revolves around objects.

Interview line: Class defines structure; object represents actual data in memory.

□ 2. Static Keyword

- Belongs to **class**, not object.
- Loaded when class loads.
- Shared across all objects.

Where used?

- static variable → shared value
- static method → utility logic
- static block → runs **once**, when class loads

Interview line: Static members belong to class memory and load once per JVM.

□ 3. Instance (Non-static)

- Belongs to **object**, not class.
- Each object gets its own copy.

Interview line: Instance members store object-specific data.

□ 4. Constructor

- Initializes object.
- Same name as class.
- No return type.

Types

- Default
- Parameterized
- Copy constructor (user-defined)

Interview line: Constructor prepares object with initial state.

□ 5. Constructor Overloading

- Multiple constructors with different parameters.
- Used for flexible initialization.

□ 6. this & super

- **this** → current class instance.
- **super** → parent class instance.

Uses

- call variable
- call method
- call constructor

□ 7. Blocks

Static Block

- Runs once when class loads.
- Used for static initialization.

Instance Block

- Runs before constructor.
- Used for common object initialization.

Execution Order:

```
Static block → Instance block → Constructor
```

□ 8. Inheritance

- Parent → Child relationship
- Code reuse / overriding

Types: Single, Multilevel, Hierarchical
(Java does NOT support multiple inheritance via classes)

□ 9. Method Overloading vs Overriding

Overloading

- Same name, different parameters
- Compile-time

Overriding

- Same name, same parameters
- Run-time

Interview line: Overloading is compile-time poly; overriding is runtime poly.

□ 10. Abstraction & Encapsulation

- **Abstraction** → hide internal logic (interface/abstract class)
- **Encapsulation** → wrap data + methods (private fields + getters/setters)

□ SPRING CORE

□ 1. IoC (Inversion of Control)

- Object creation control → developer → Spring
- Spring manages beans

Interview line: IoC shifts object creation responsibility to Spring.

□ 2. Dependency Injection (DI)

- Inject required objects
- Reduces tight coupling

Types

- Constructor injection (best)
- Setter injection
- Field injection (not recommended)

□ 3. @Component Family

- @Component → general bean
- @Service → business logic
- @Repository → database layer + exception translation
- @Controller → MVC controller
- @RestController → returns JSON

□ 4. Bean Scope

- **singleton** (default)
- **prototype**
- request/session (web only)

□ 5. Bean Lifecycle

```
Constructor → @Autowired → @PostConstruct → (use) → @PreDestroy
```

□ 6. @Autowired + @Qualifier + @Primary

- @Autowired → inject bean
- @Qualifier → select specific bean
- @Primary → default bean to choose

□ 7. @Configuration vs @Bean vs @Component

- @Bean → manual bean creation (3rd-party classes)

- @Component → auto-detected bean
- @Configuration → holds @Bean methods

□ SPRING MVC (REST API)

□ 1. @Controller vs @RestController

- @Controller → HTML views
- @RestController → JSON REST API

□ 2. REST Annotations

- @GetMapping → read
- @PostMapping → create
- @PutMapping → update
- @DeleteMapping → delete

Input handling

- @RequestBody → JSON → Object
- @RequestParam → query param
- @PathVariable → URL variable

□ 3. DispatcherServlet

- Front controller
- Handles request → maps to controller → returns response

□ REACT JS FUNDAMENTALS

□ 1. Component

- Function returning JSX
- Building block of UI

□ 2. JSX

- HTML-like syntax inside JS
- Compiled to React.createElement

□ 3. Props

- Read-only inputs to components
- Passed parent → child

Interview line: Props are immutable component inputs.

□ 4. State (useState)

- Component's internal memory
- Updating state re-renders UI

Syntax

```
const [value, setValue] = useState(initial);
```

□ 5. Event Handling

```
<button onClick={fn}>Click</button>
```

□ 6. Conditional Rendering

```
{condition && <Component />}
```

□ 7. Lists & Keys

```
items.map(item => <li key={item.id}>{item.name}</li>)
```

□ 8. useEffect (basic intro)

- Handles side effects (fetch, timers)

```
useEffect(() => { ... }, []);
```

□ 9. Controlled Components (forms)

```
<input value={value} onChange={e => setValue(e.target.value)} />
```

□ 10. Component Communication

- Parent → child = props
- Child → parent = callback functions

□ FINAL INTERVIEW REMINDERS

- Java → Focus: OOP, static, memory, overriding, interfaces
- Spring → Focus: IoC, DI, lifecycle, MVC, REST
- React → Focus: components, props, state, hooks