Java Backend Developer Complete Notes

----

1. Java Basics

What is Java?

- Java is a high-level, object-oriented, platform-independent programming language.
- Write once, run anywhere (WORA) because Java compiles to bytecode which runs on the Java Virtual Machine (JVM).

Key Points:

- Statically typed: You declare data types.
- Compiled & interpreted: Compiled to bytecode, interpreted by JVM.
- Strong standard library.

Example Hello World:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

----

2. OOPs in Java (Detailed)

OOPs Principles:

- Encapsulation: Bundling data & methods that work on data in one unit.
- Inheritance: Acquiring properties from a parent class.
- Polymorphism: One interface, many implementations (overloading & overriding).
- Abstraction: Hiding implementation details & exposing only necessary parts.

Examples:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}

public class Test {
    public static void main(String args[]) {
        Animal a = new Dog();
        a.sound();
    }
}
```

----

3. Collection Framework (Detailed with Visualization)

Main interfaces:

- List (ArrayList, LinkedList)
- Set (HashSet, TreeSet)
- Queue (LinkedList, PriorityQueue)
- Map (HashMap, TreeMap)

Visualization:

```
List: [10] -> [20] -> [30]
Set: {10, 20, 30} (no duplicates, unordered)
Map: { "key1": "value1", "key2": "value2" }
```

Example:

```
List<String> list = new ArrayList<>();
list.add("apple");
list.add("banana");

Set<String> set = new HashSet<>();
set.add("apple");

Map<String, Integer> map = new HashMap<>();
map.put("apple", 10);
```

----

4. Decorator Pattern (Detailed)

- Structural design pattern to add responsibilities dynamically.

Example:

```
interface Coffee {
    String makeCoffee();
}

class SimpleCoffee implements Coffee {
    public String makeCoffee() {
        return "Simple Coffee";
    }
}

class MilkDecorator implements Coffee {
    private Coffee coffee;
    public MilkDecorator(Coffee c) {
        this.coffee = c;
    }
    public String makeCoffee() {
        return coffee.makeCoffee() + " + Milk";
    }
}
```

---

5. Spring (Short)

---

- Framework for enterprise Java applications.
- Handles dependency injection (DI), transaction management, AOP.

----

6. Spring Boot (Detailed)

- Built on Spring, simplifies microservices & REST API development.
- Embedded server (Tomcat), auto-configuration.

Main files:

- @SpringBootApplication: entry point.
- application.properties: configurations.

----

7. REST API (GET, POST, PUT, PATCH, DELETE)

```
@RestController
@RequestMapping("/api/items")
public class ItemController {
   @GetMapping public List<Item> getAll() { ... }
   @PostMapping public Item create(@RequestBody Item i) { ... }
   @PutMapping("/{id}") public Item update(@PathVariable Long id, @RequestBody Item i) { ... }
       @PatchMapping("/{id}") public Item partialUpdate(@PathVariable Long id, @RequestBody
Map<String, Object> updates) { ... }
   @DeleteMapping("/{id}") public void delete(@PathVariable Long id) { ... }
}
```