🧱 1. Singleton Pattern

🔹 JDK Internal Example

```
Runtime runtime = Runtime.getRuntime();
```

- `Runtime` class uses the Singleton pattern.
- It restricts instantiation using a private constructor and exposes a static `getRuntime()` method.

🔹 Spring Boot Example

```
@Service
public class AuditLogger {
    public void log(String msg) {
        System.out.println("AUDIT: " + msg);
    }
}
```

- In Spring, all beans annotated with `@Component`, `@Service`, `@Repository`, etc., are **Singletons by default** in the ApplicationContext.

🏭 2. Factory Method Pattern

🔹 JDK Internal Example

```
List<String> list = List.of("a", "b");
Set<String> set = Set.of("x", "y");
```

- The `List.of()` and `Set.of()` methods are **static factory methods** introduced in Java 9.
- Internally, based on arguments, they return different internal implementations (e.g., `ImmutableCollections.List12`, etc.).

🔹 Spring Boot Example

```
@Bean
public DataSource dataSource() {
    return DataSourceBuilder.create()
            .url("jdbc:mysql://localhost/db")
            .username("root")
            .password("secret")
            .build();
}
```

- `DataSourceBuilder.create()` uses the factory method pattern.

- Spring Boot itself uses `FactoryBean<T>` interface for pluggable object creation (e.g., `LocalContainerEntityManagerFactoryBean`).

🧰 3. Abstract Factory Pattern

🔹 JDK Internal Example

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

- `DocumentBuilderFactory` and `TransformerFactory` are abstract factories.
- They return concrete implementations of families of related XML parsers/builders depending on platform/vendor.

🔹 Spring Boot Example

```
@Configuration
public class MessagingConfiguration {

    @Bean
    @Profile("kafka")
    public MessageClient kafkaClient() {
        return new KafkaClient();
    }

    @Bean
    @Profile("rabbitmq")
    public MessageClient rabbitMqClient() {
        return new RabbitMqClient();
    }
}
```

- This acts like an **abstract factory**: Spring provides different `MessageClient` implementations for different profiles (families of related components).
- Spring's `ApplicationContext` itself can be considered an abstract factory.

🏗️ 4. Builder Pattern

🔹 JDK Internal Example

```
StringBuilder sb = new StringBuilder();
sb.append("Hello").append(" ").append("World");
```

- `StringBuilder` is the classic builder: it constructs a complex `String` in steps.

Also:

```
HttpRequest request = HttpRequest.newBuilder()
        .uri(new URI("http://example.com"))
        .header("User-Agent", "Java 11")
        .GET()
        .build();
```

- `HttpRequest.Builder` from `java.net.http` module is a **pure builder pattern**.

🔹 Spring Boot Example

```
SecurityFilterChain filterChain = http
    .authorizeHttpRequests(auth -> auth
        .requestMatchers("/admin").hasRole("ADMIN")
        .anyRequest().authenticated()
    )
    .httpBasic(Customizer.withDefaults())
    .build();
```

- Spring Security DSL (`http`) is a fluent **Builder** API for configuring filters and security rules.

Also:

```
DataSource ds = DataSourceBuilder.create()
    .url("jdbc:h2:mem:testdb")
    .username("sa")
    .build();
```

🧬 5. Prototype Pattern

🔹 JDK Internal Example

```
ArrayList<String> original = new ArrayList<>();
ArrayList<String> clone = (ArrayList<String>) original.clone();
```

- `clone()` method from `Object` class demonstrates shallow **Prototype** pattern.
- Classes like `ArrayList`, `HashMap`, etc., implement `Cloneable`.

🔹 Spring Boot Example

```
@Scope("prototype")
@Component
public class Notification {
    // New instance created each time
}


@Autowired
private ApplicationContext context;
```

```
public void sendNotification() {
    Notification notification = context.getBean(Notification.class);
    notification.send(); // new instance each time
}
```

- Spring beans with `@Scope("prototype")` behave like **Prototype Pattern**: a new object is returned for every request.

✅ Summary Table

Pattern	JDK Internal Example	Spring Boot Example
Singleton	`Runtime.getRuntime()`	`@Service`, `@Component` beans
Factory Method	`List.of()`, `Calendar.getInstance()`	`DataSourceBuilder.create()`
Abstract Factory	`DocumentBuilderFactory.newInstance()`	Profile-based bean factories, `ApplicationContext`
Builder	`StringBuilder`, `HttpRequest.Builder`	`SecurityFilterChain`, `DataSourceBuilder`, Fluent APIs
Prototype	`ArrayList.clone()`, `Object.clone()`	`@Scope("prototype")` beans with `ApplicationContext.getBean()`

✅ Summary Table

Pattern	Core Idea	Spring Boot Usage
Singleton	One global instance	`@Service`, `@Component` beans are singletons by default
Factory Method	Subclass decides what to instantiate	Component-based factory + switch
Abstract Factory	Create families of related objects	Conditional injection or `@Profile`, factory beans
Builder	Step-by-step object construction	Builder pattern used in DTOs or fluent config objects
Prototype	Clone existing object	Custom registry + cloning per request

---