Absolutely! Let's break this down into **useful tips for Spring Boot, Maven, Java**, plus how you can make a **user-friendly EXE for your Spring Boot app**. I'll go step by step, practical, and beginner-friendly but also professional.

---

1. **Tips for Spring Boot**

Spring Boot simplifies Java backend development. Here are some key tips:

a) Project structure

- Keep standard Maven structure:

```
src/main/java       -> your Java code
src/main/resources  -> application.properties, templates
src/test/java       -> test code
```

- Follow **package naming conventions**: com.company.project

b) Application.properties / yml

- Central place for configuration (DB, server port, logging, etc.)

```
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=root
spring.datasource.password=1234
```

- Use application-dev.properties and application-prod.properties for environments.

c) Spring Profiles

- Useful to switch configurations between dev, test, prod:

```
@Profile("dev")
@Service
public class DevService { ... }
```

- Run with profile:

```
java -jar myapp.jar --spring.profiles.active=dev
```

d) Embedded Tomcat

- No need to install Tomcat separately; Spring Boot runs on **embedded Tomcat** by default.

e) REST API tips

- Use @RestController for APIs.
- Use @RequestMapping or @GetMapping, @PostMapping.
- Use **DTOs** for input/output instead of entities directly.

---

2. ips for Maven (mvn)

Maven is the **build and dependency management tool** for Java projects.

a) Common commands

Command	Use
mvn clean	Remove target folder
mvn compile	Compile Java code
mvn test	Run unit tests
mvn package	Build JAR/WAR
mvn install	Install into local Maven repo
mvn spring-boot:run	Run Spring Boot app directly

b) Dependency management

- Add dependencies in pom.xml:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- Maven downloads them automatically.

c) Plugins

- Spring Boot Maven plugin helps build **executable JARs**:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

3. Tips for Java

a) Java versions

- Use **Java 17+** for Spring Boot 3.x
- New features:
    - var keyword for local variable type inference
    - Records (record Person(String name, int age))
    - Pattern matching for instanceof
    - Text blocks for multiline strings (""" ... """)

b) Exception handling

- Always handle exceptions gracefully in REST APIs:

```
@RestControllerAdvice
public class GlobalExceptionHandler { ... }
```

c) Unit testing

- Use **JUnit 5** for tests.
- Use **Mockito** for mocking services.
- Run tests: mvn test

4. How to make a Windows EXE with UI for user access

Spring Boot is **backend only**, so for a UI you need one of these approaches:

Option 1: Web-based UI

- Build a **frontend with HTML/JS/React/Vue**
- Package it with Spring Boot using src/main/resources/static
- Run the app and open in browser → "UI for user"
- Then create an EXE installer for Spring Boot backend using jpackage or app-image
- User double-click EXE → opens backend → frontend served via browser

Option 2: Java Desktop UI (Swing / JavaFX)

- Build **JavaFX or Swing GUI** in the same project
- Call your Spring Boot services via HTTP or directly
- Then package with **jpackage** to make a **single EXE**

Example:

```
MyApp/
  src/main/java/com/example/gui/MainUI.java  --> JavaFX window
  src/main/java/com/example/demo/DemoApplication.java --> Spring Boot backend
```

- JavaFX runs GUI → calls Spring Boot APIs in the same JVM
- Use jpackage --type exe to create EXE with GUI

---

5. Recommended approach for modern apps

- **Web UI + Spring Boot backend** is easiest:
  - Build frontend with React/Vue/Angular
  - Serve static files from Spring Boot
  - Use jpackage to wrap Spring Boot backend as EXE (user double-click → app runs → opens browser automatically)
- **Desktop Java UI** is less common today, unless it's an internal tool.

---

6. Example: EXE + Web UI approach

- Dockerfile or jpackage EXE contains:
  - Spring Boot backend
  - React frontend in src/main/resources/static
- EXE startup opens browser automatically:

```
public static void main(String[] args) {
   SpringApplication.run(MyApplication.class, args);
   try {
      java.awt.Desktop.getDesktop().browse(new URI("http://localhost:8080"));
   } catch (Exception e) { e.printStackTrace(); }
}
```

- User experience: **double-click EXE → app runs → browser opens automatically with UI**