EXPLANATION OF CODING

Here's an explanation of code using the **DRY** principle:

1. Code Without DRY Principle (Repetitive Code)

In this version, the code has multiple methods for greeting in different languages. Each method has similar functionality (printing a greeting) but is repeated for each language.

java

```
public class Greeting {

  // Method to greet in English
  public static void greetInEnglish() {
    System.out.println("Hello!");
  }

  // Method to greet in Spanish
  public static void greetInSpanish() {
    System.out.println("¡Hola!");
  }

  // Method to greet in French
  public static void greetInFrench() {
    System.out.println("Bonjour!");
  }

  public static void main(String[] args) {
    // Calling individual methods
    greetInEnglish();
```

```
    greetInSpanish();
    greetInFrench();
  }
}
```

Drawbacks of Repetitive Code:

- **Repetition:** Similar logic (printing a greeting) is repeated in each method.
- **Low maintainability:** Adding new languages requires creating new methods, which increases code duplication.
- **Harder to extend:** If more greetings are needed (e.g., Italian, German), the program becomes bloated with more methods, reducing scalability.

---

2. Code With DRY Principle (Generalized Code)

The DRY principle is applied by creating a **single method** that handles greeting in multiple languages. This reduces redundancy and simplifies the code.

java

Copy code

```
public class Greeting {

  // Generalized method to greet in different languages
  public static void greet(String language) {
    switch (language.toLowerCase()) {
      case "english":
        System.out.println("Hello!");
        break;
      case "spanish":
        System.out.println("¡Hola!");
        break;
      case "french":
```

```
            System.out.println("Bonjour!");
            break;
        default:
            System.out.println("Language not supported");
    }
  }

  public static void main(String[] args) {
    // Calling the generalized method
    greet("english");
    greet("spanish");
    greet("french");
  }
}
```

---

3. Advantages of Using DRY Principle

By applying the DRY principle, the code becomes more efficient and easier to maintain. Below are the key advantages:

3.1. Reusability

- **One Method for All Languages:** The greet method is designed to handle all languages by using a parameter (language) to decide which greeting to print.
- **No Duplication:** The logic of printing a greeting is written only once, and the method is reused for different languages.

3.2. Maintainability

- **Easier Updates:** If a greeting needs to be changed (e.g., modifying the French greeting), it can be updated in a single location inside the switch statement.
- **Adding New Languages:** To add support for new languages, such as Italian, you only need to add one more case to the existing switch statement, rather than writing a whole new method.

3.3. Scalability

- **Simple Expansion:** The code is more scalable since adding more greetings involves updating just one method rather than multiple individual methods.
- **Cleaner Structure:** The use of a single method leads to cleaner and more organized code, making it easier to manage as the application grows.

---

4. Conclusion: Benefits of DRY in This Example

By applying the **DRY principle** in the second version of the code, we reduce complexity and duplication. The code becomes:

- **Simpler to read and maintain.**
- **Easier to extend** by adding new languages.
- **More efficient** by consolidating logic into a single, reusable method.

This approach minimizes code repetition, making it easier to handle changes and improvements in the future.