Java 21 Features

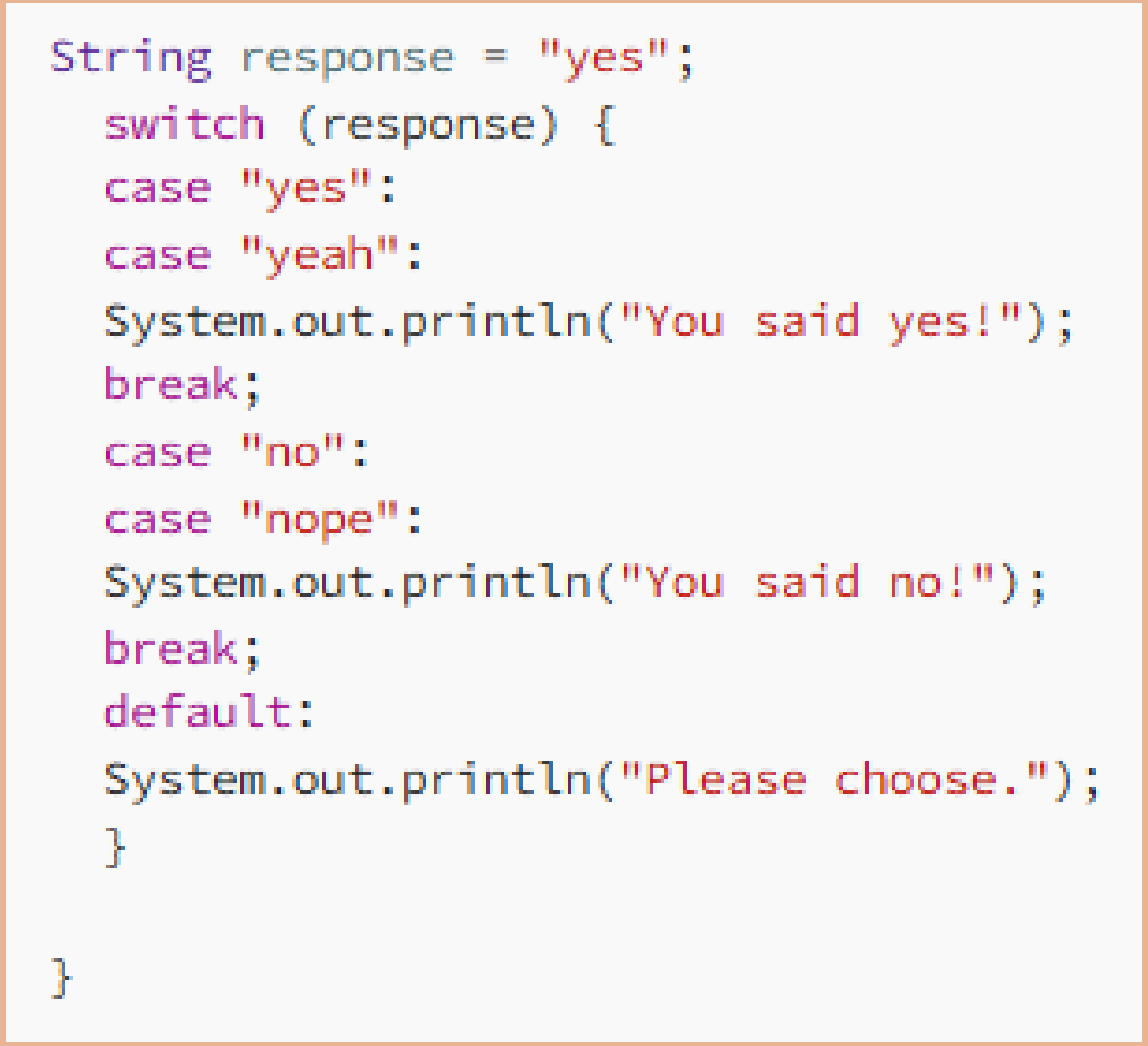
Swipe Left →

Java 21 brings some exciting new features to the world of programming. In this article, we’ll explore these Java 21 features with practical examples to make your Java coding experience even better.

Pattern Matching for Switch:

Java 21 brings a powerful feature called Pattern Matching for Switch. It simplifies switch statements, making them more concise and readable. Check out an example:

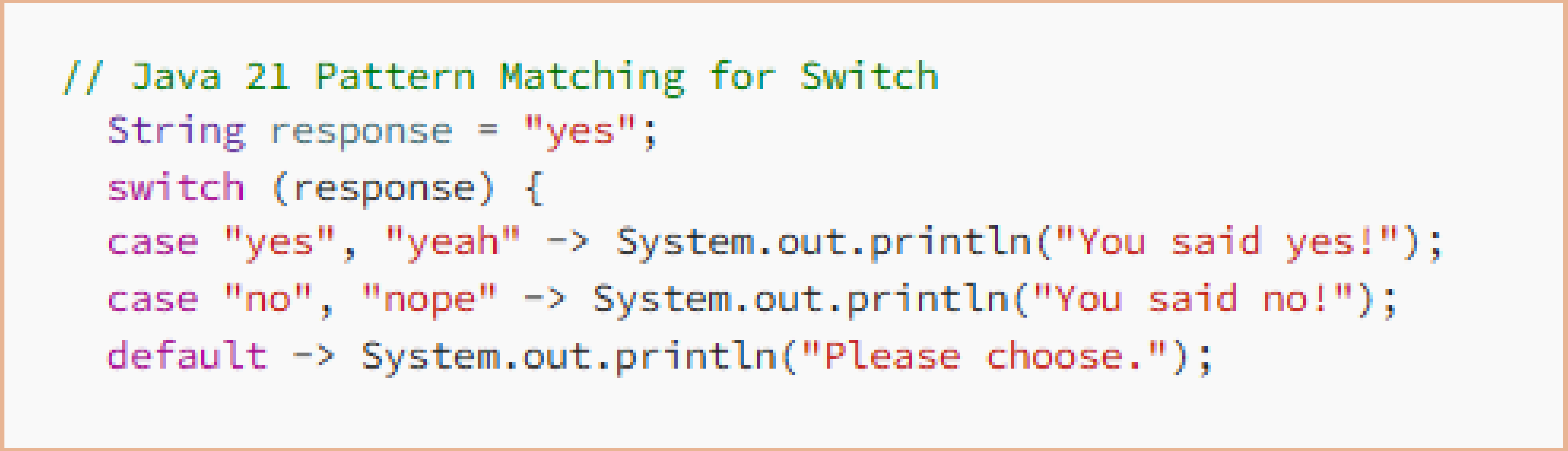
// Before Java 21 :



```
String response = "yes";
  switch (response) {
  case "yes":
  case "yeah":
  System.out.println("You said yes!");
  break;
  case "no":
  case "nope":
  System.out.println("You said no!");
  break;
  default:
  System.out.println("Please choose.");
  }

}
```

Pattern Matching for Switch in java 21:



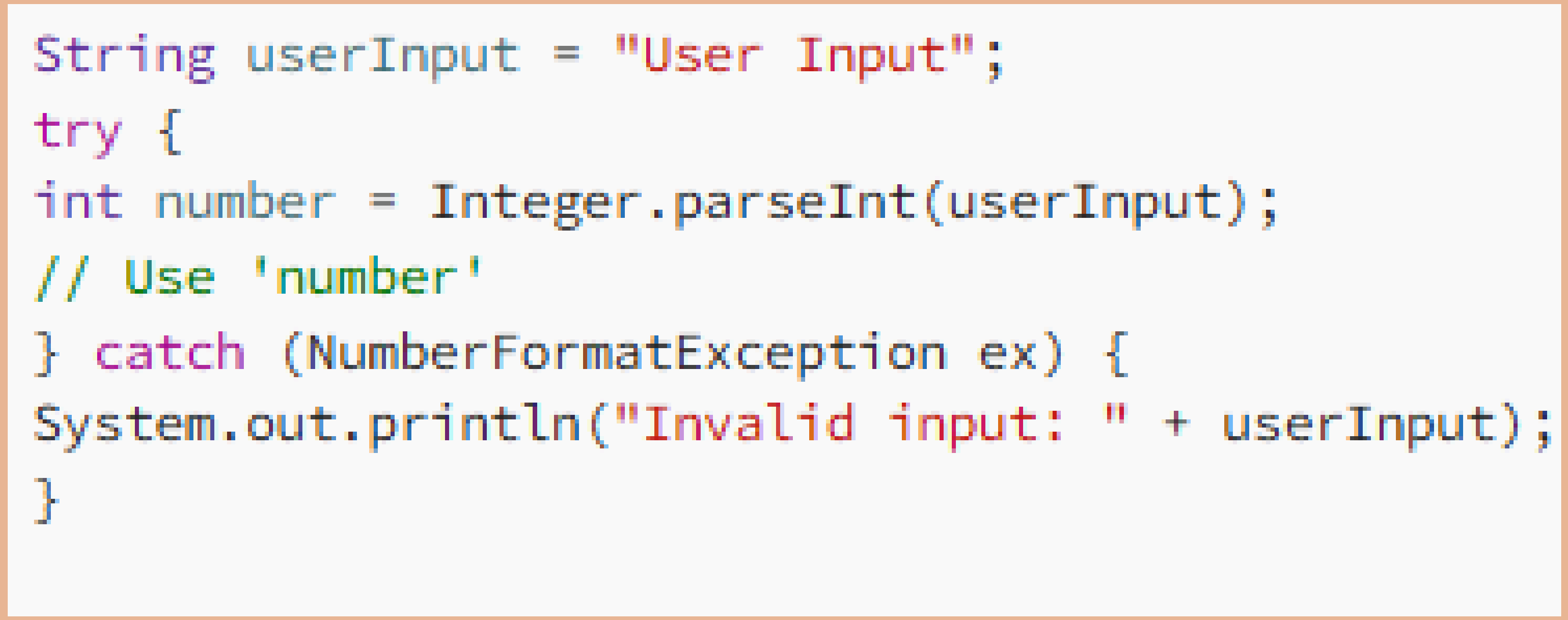
```
// Java 21 Pattern Matching for Switch
  String response = "yes";
  switch (response) {
  case "yes", "yeah" -> System.out.println("You said yes!");
  case "no", "nope" -> System.out.println("You said no!");
  default -> System.out.println("Please choose.");
```

Benefits:

- Eliminates verbose instanceof checks
- Brings better type safety
- Improves code readability and maintainability

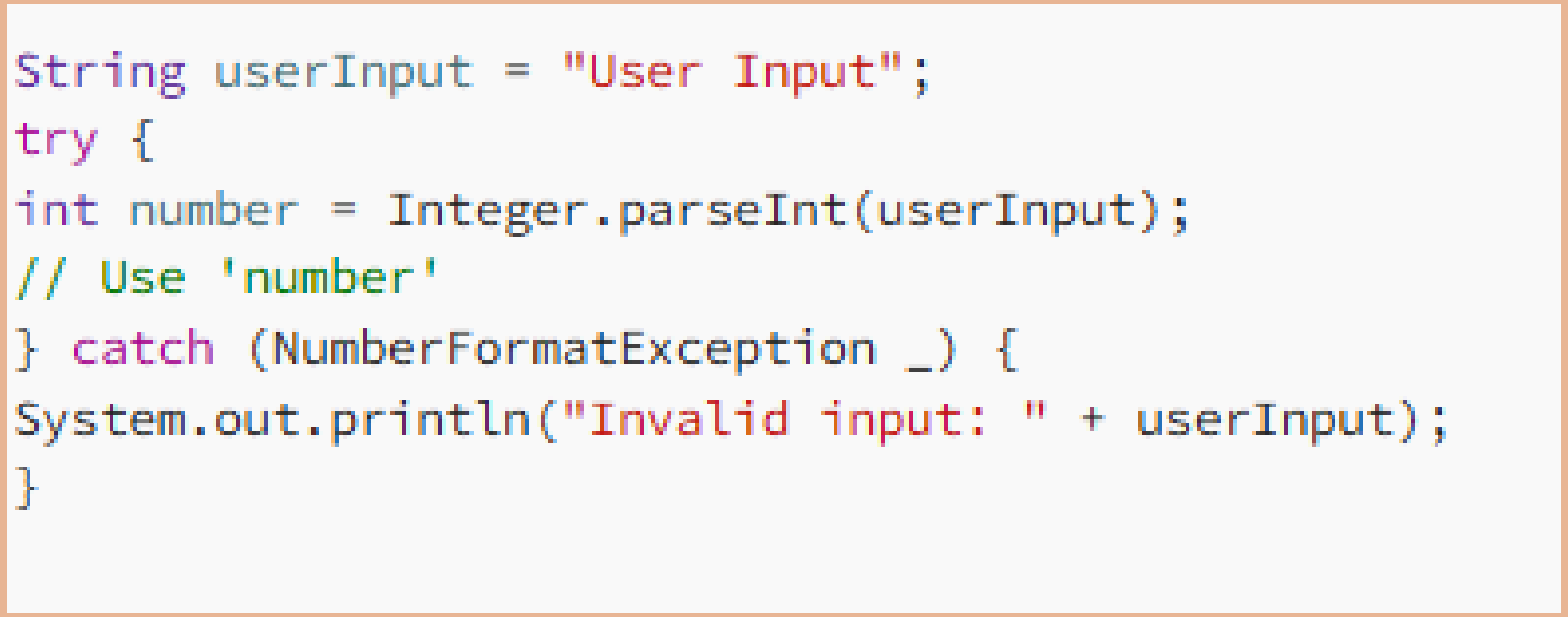
Unnamed Patterns and Variables

Java 21 introduces Unnamed Patterns and Variables, making your code more concise and expressive. Here's a snippet to give you a taste:



```
String userInput = "User Input";
try {
int number = Integer.parseInt(userInput);
// Use 'number'
} catch (NumberFormatException ex) {
System.out.println("Invalid input: " + userInput);
}
```

=>



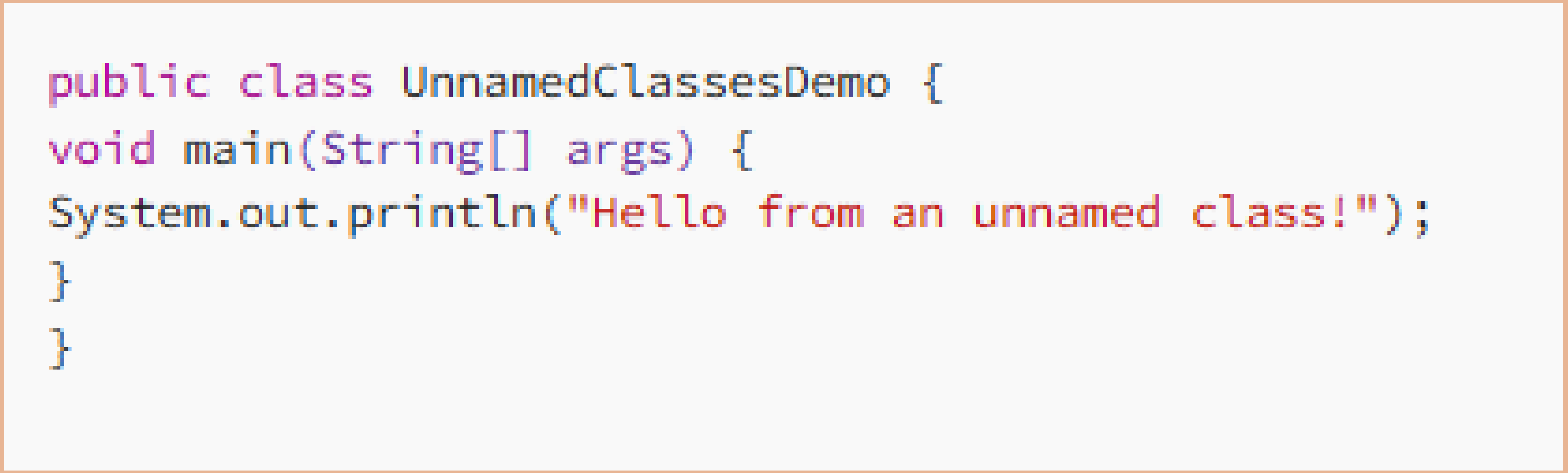
```
String userInput = "User Input";
try {
int number = Integer.parseInt(userInput);
// Use 'number'
} catch (NumberFormatException _) {
System.out.println("Invalid input: " + userInput);
}
```

In this updated version, we no longer use the 'ex' variable; instead, we've replaced it with an underscore (_). This simple change helps streamline the code and makes it more concise.

Unnamed Classes and Instance Main Methods:

Java 21 introduces a fresh approach to defining classes and instance main methods right in your code. Let’s take a quick look at how this feature operates:

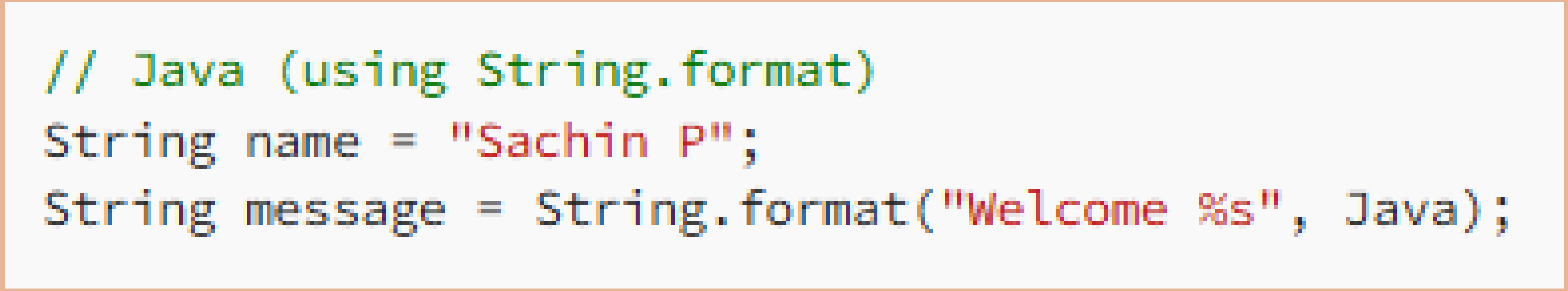
// Java 21 Unnamed Classes and Instance Main Methods



```
public class UnnamedClassesDemo {
void main(String[] args) {
System.out.println("Hello from an unnamed class!");
}
}
```

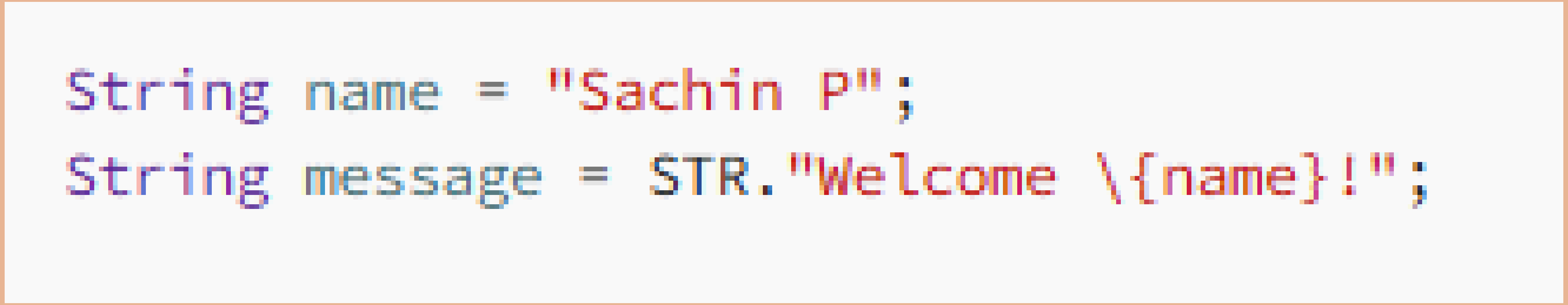
String Templates in Java

Java 21 introduces String Templates, simplifying string concatenation. Take a look:



```java
// Java (using String.format)
String name = "Sachin P";
String message = String.format("Welcome %s", Java);
```

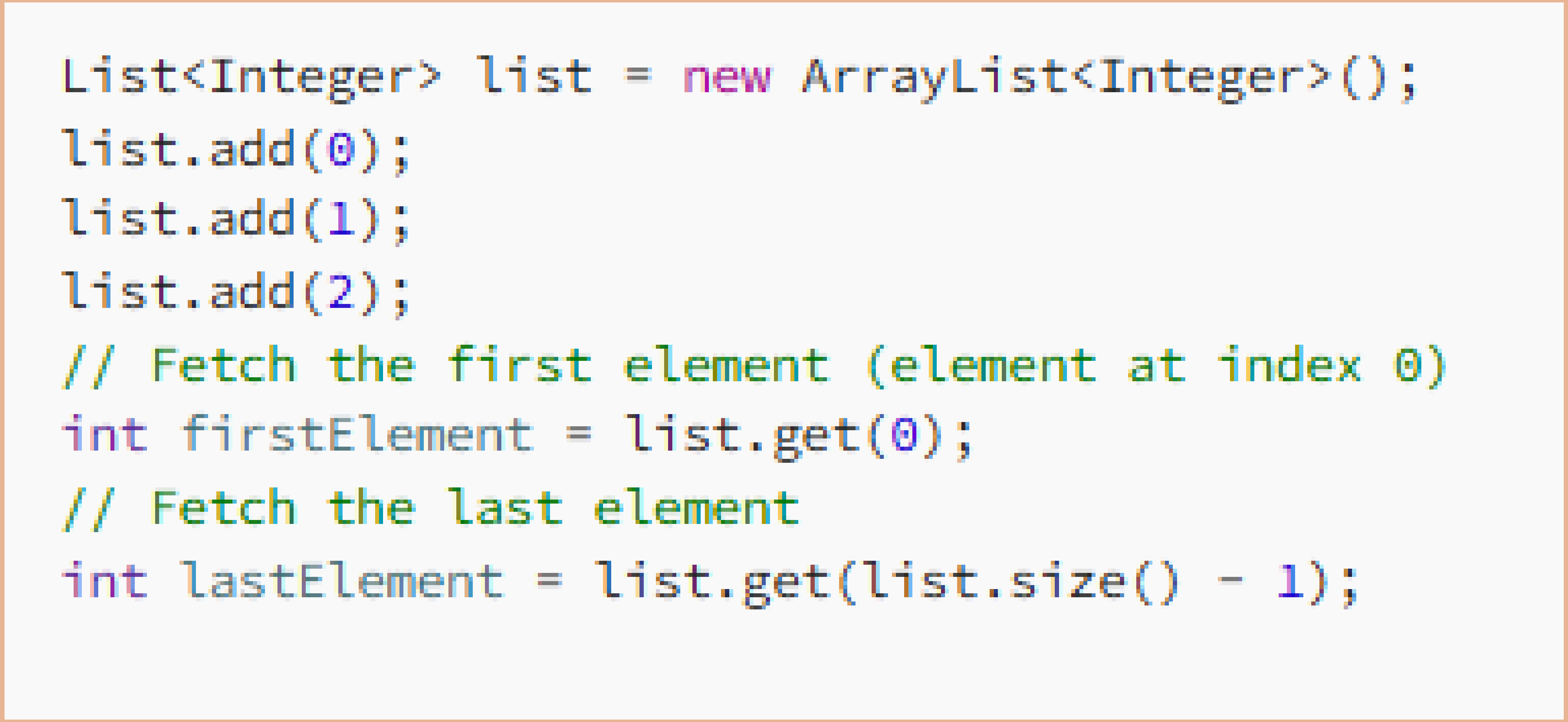
In Java 21, you can create a message using this syntax:



```java
String name = "Sachin P";
String message = STR."Welcome \{name}!";
```

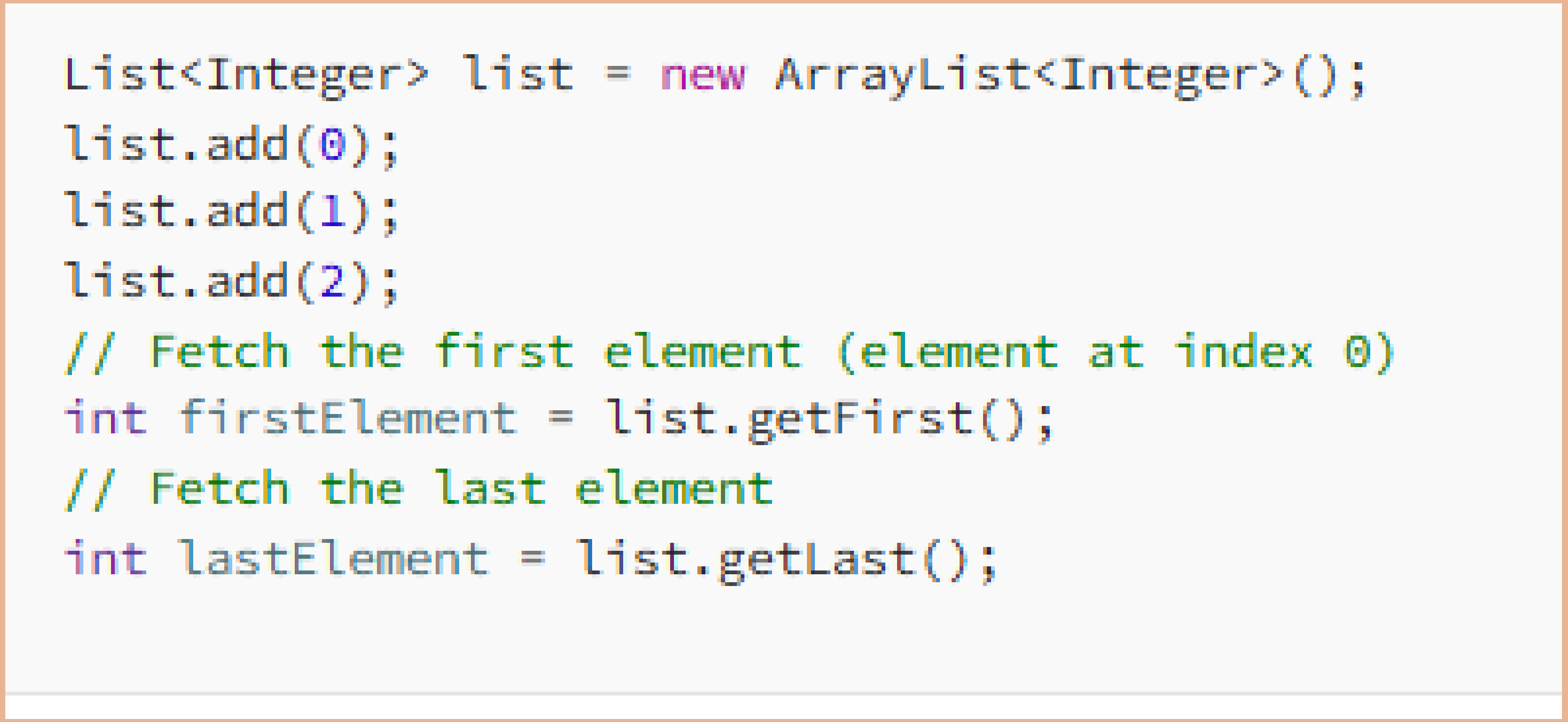
Sequenced Collections in Java 21

Java 21 introduces Sequenced Collections, making it easier to work with ordered data. Here’s a glimpse:



```java
List<Integer> list = new ArrayList<Integer>();
list.add(0);
list.add(1);
list.add(2);
// Fetch the first element (element at index 0)
int firstElement = list.get(0);
// Fetch the last element
int lastElement = list.get(list.size() - 1);
```

In Java 21, you can retrieve elements using the following code.



```java
List<Integer> list = new ArrayList<Integer>();
list.add(0);
list.add(1);
list.add(2);
// Fetch the first element (element at index 0)
int firstElement = list.getFirst();
// Fetch the last element
int lastElement = list.getLast();
```

Why it matters:

- Brings consistency across collection types
- Simplifies logic when dealing with head/tail of collections
- Improves code readability and reduces boilerplate

Conclusion

Java 21 brings modern enhancements like Pattern Matching for switch, Sequenced Collections for ordered data handling, Record Patterns, and Virtual Threads for scalable concurrency. It also introduces String Templates (preview) and Scoped Values. These updates boost code clarity, performance, and developer productivity.