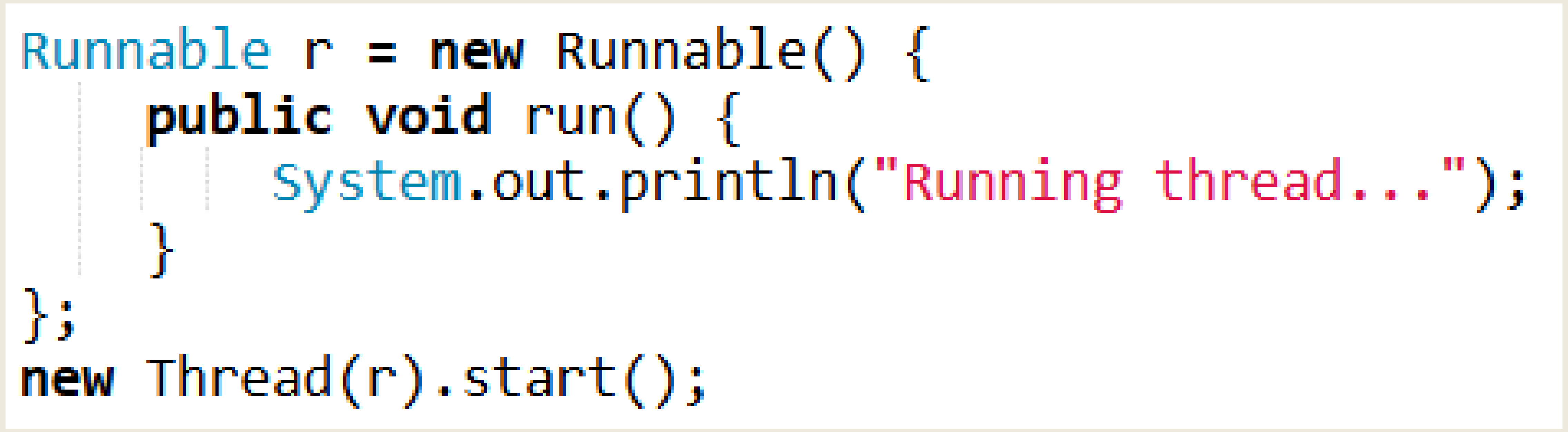
Java 8 Features Explained with Real Examples – Write Cleaner and Smarter Code!

Swipe Left →

1. Lambda Expressions

Before Java 8:

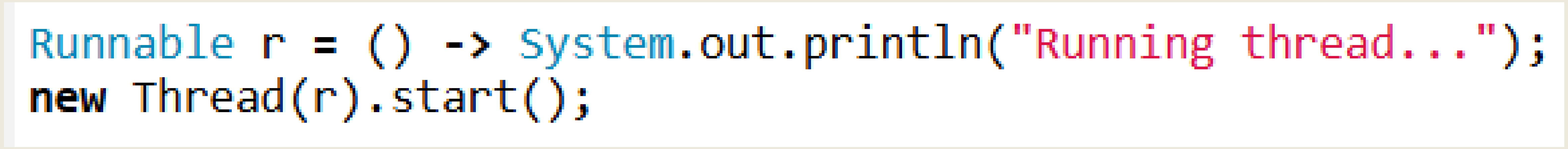
You had to use anonymous classes for simple implementations.



```
Runnable r = new Runnable() {
    public void run() {
        System.out.println("Running thread...");
    }
};
new Thread(r).start();
```

With Java 8:

Lambda expressions make it concise.



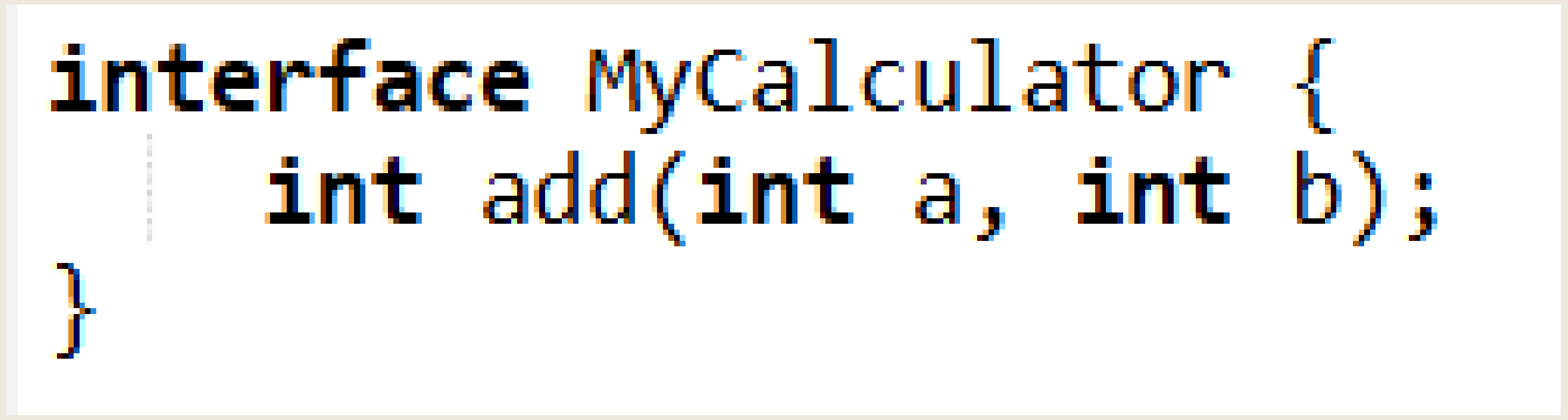
```
Runnable r = () -> System.out.println("Running thread...");
new Thread(r).start();
```

2. Functional Interface

A functional interface has only one abstract method. Java 8 introduced @FunctionalInterface annotation.

Before Java 8:

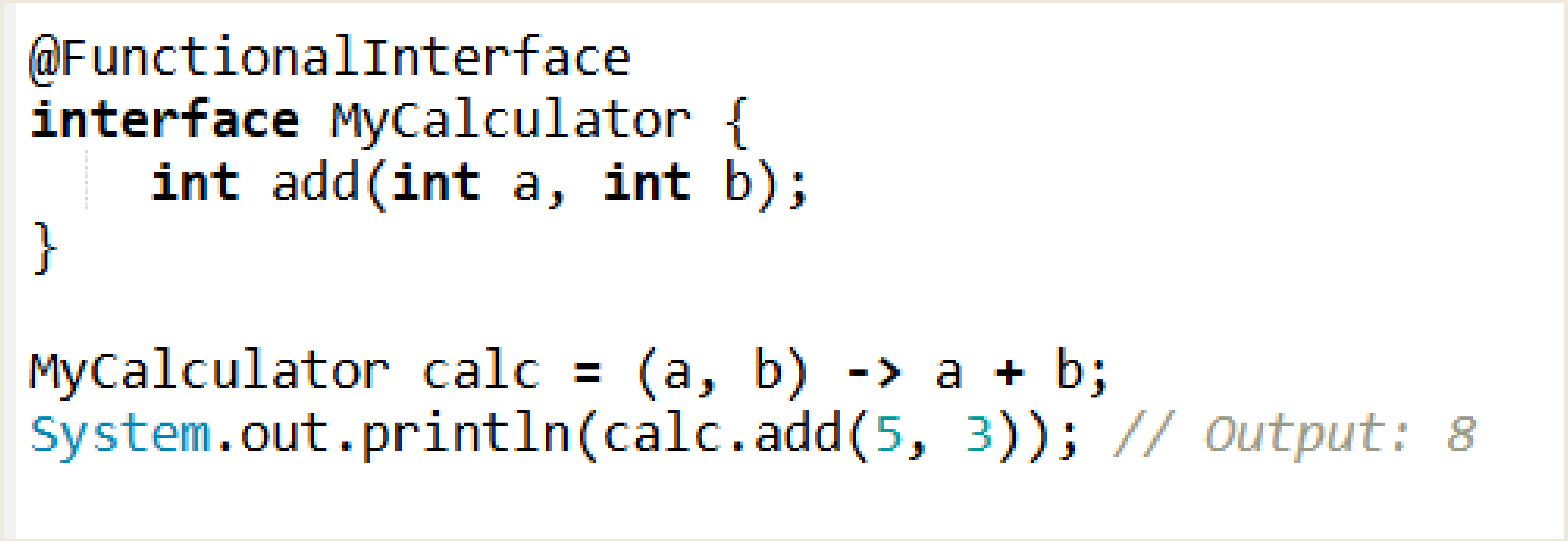
You would write interfaces and implement them using classes or anonymous classes.



```
interface MyCalculator {
    int add(int a, int b);
}
```

With Java 8:

Use the interface with lambda expressions.

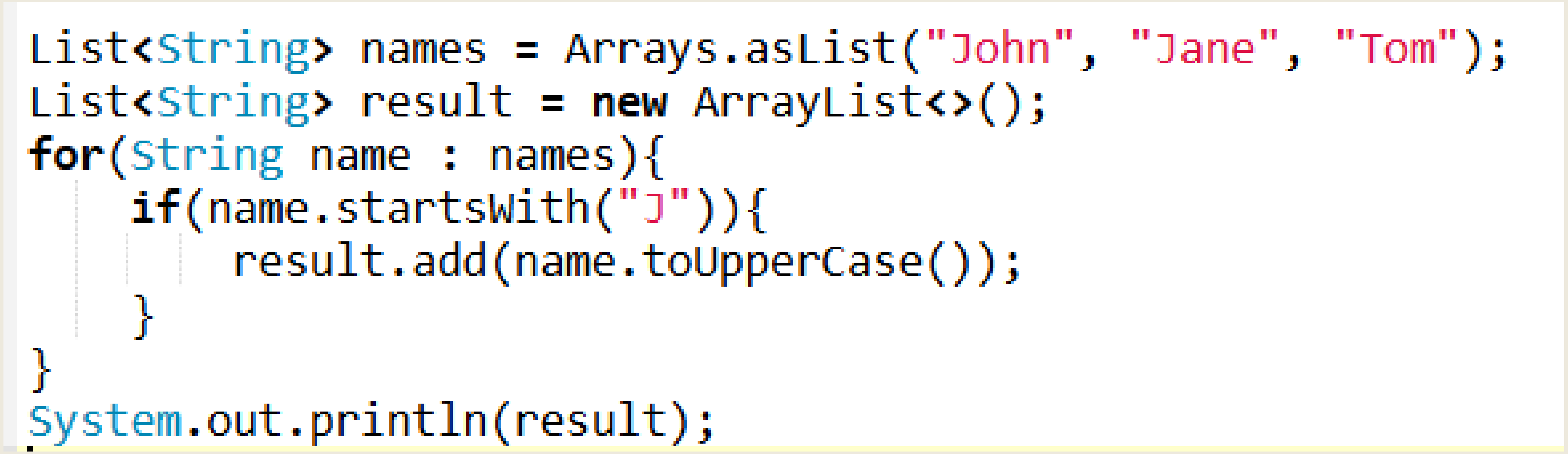


```java
@FunctionalInterface
interface MyCalculator {
    int add(int a, int b);
}

MyCalculator calc = (a, b) -> a + b;
System.out.println(calc.add(5, 3)); // Output: 8
```

3. Stream API

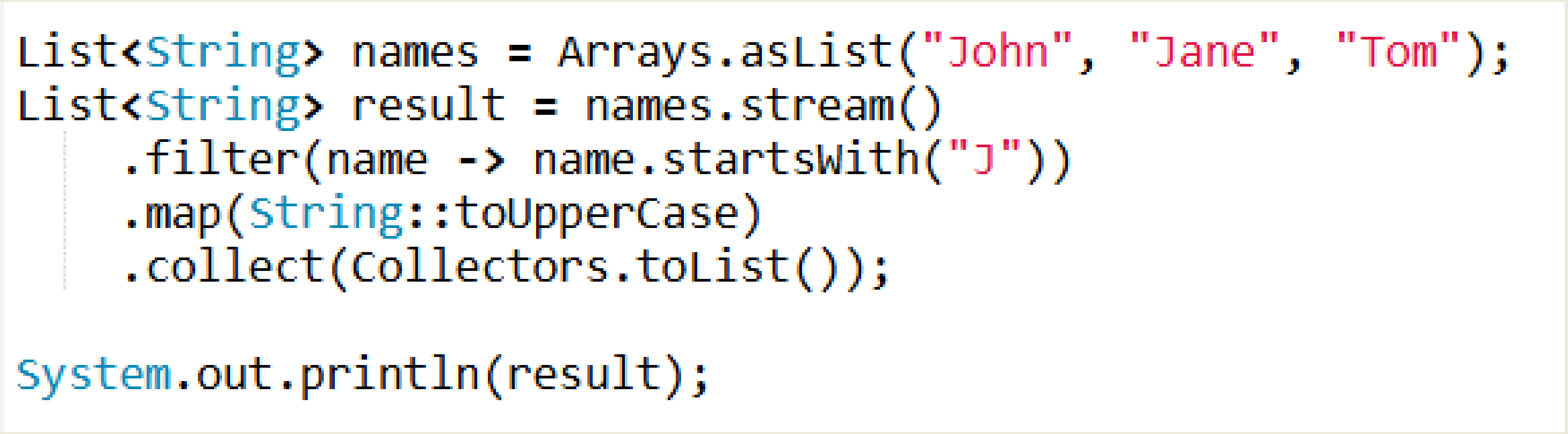
Stream API helps process collections in a functional style.

Before Java 8:



```
List<String> names = Arrays.asList("John", "Jane", "Tom");
List<String> result = new ArrayList<>();
for(String name : names){
    if(name.startsWith("J")){
        result.add(name.toUpperCase());
    }
}
System.out.println(result);
```

With Java 8:

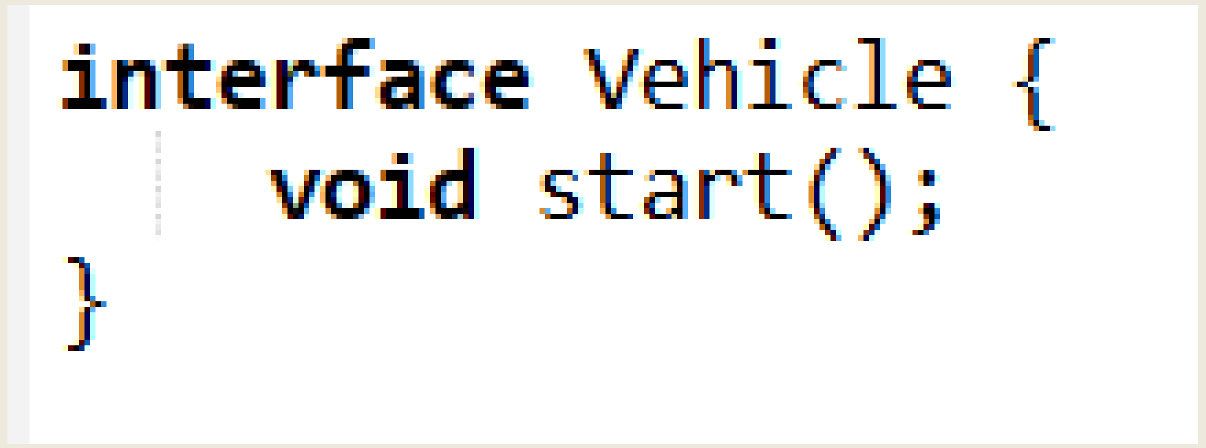


```java
List<String> names = Arrays.asList("John", "Jane", "Tom");
List<String> result = names.stream()
    .filter(name -> name.startsWith("J"))
    .map(String::toUpperCase)
    .collect(Collectors.toList());

System.out.println(result);
```

4. Default and Static Methods in Interfaces

Before Java 8:

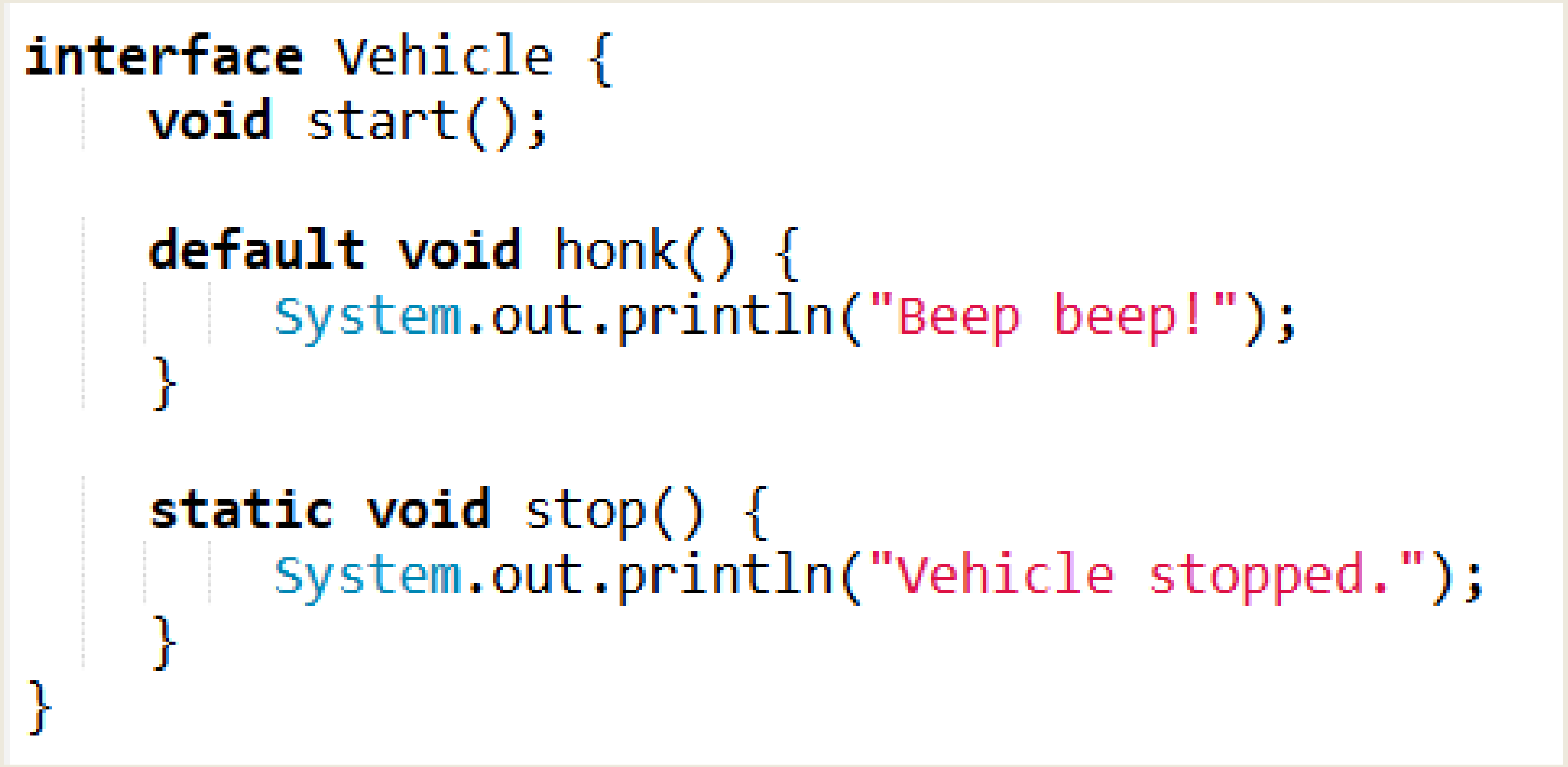
Interfaces couldn’t have method implementations.



```java
interface Vehicle {
    void start();
}
```

With Java 8:

You can have default and static methods.

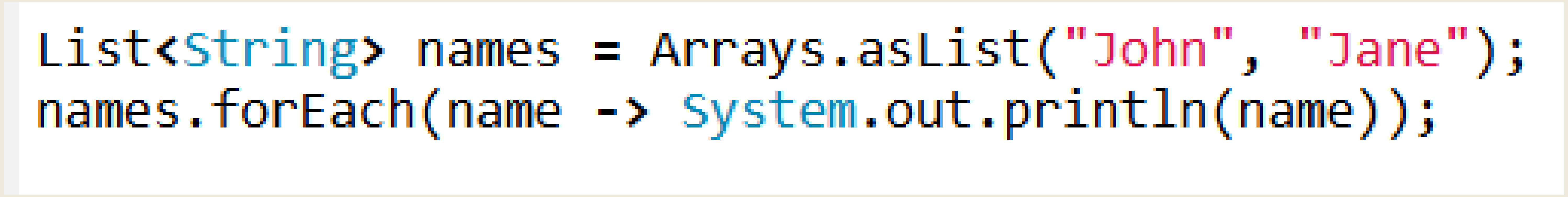


```
interface Vehicle {
    void start();

    default void honk() {
        System.out.println("Beep beep!");
    }

    static void stop() {
        System.out.println("Vehicle stopped.");
    }
}
```

5. Method References

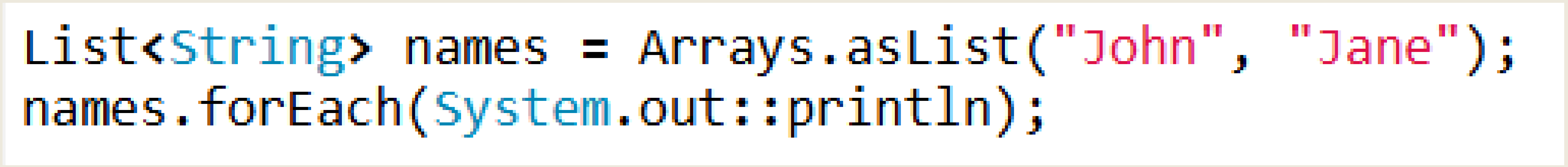
Short-hand syntax for calling methods using :: .

Before Java 8:



```java
List<String> names = Arrays.asList("John", "Jane");
names.forEach(name -> System.out.println(name));
```

With Java 8:

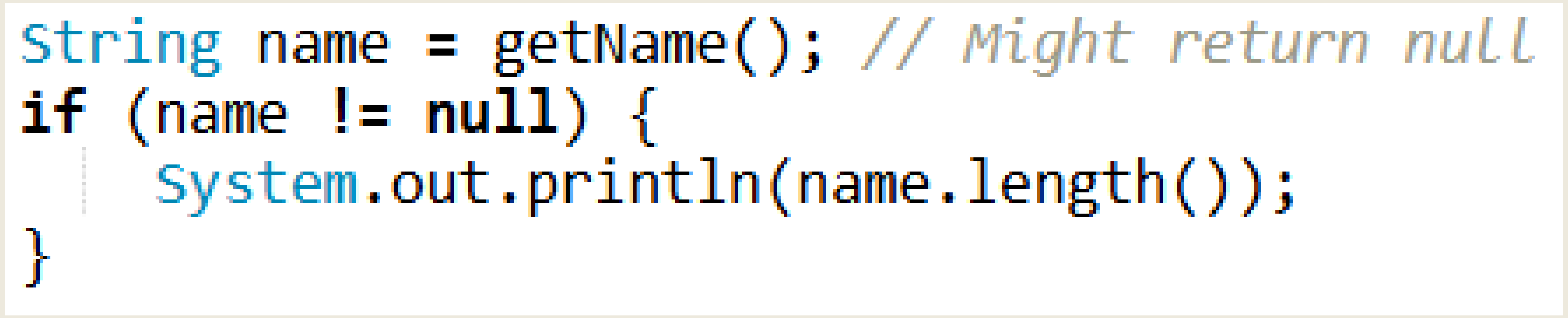


```java
List<String> names = Arrays.asList("John", "Jane");
names.forEach(System.out::println);
```

6. Optional<T>

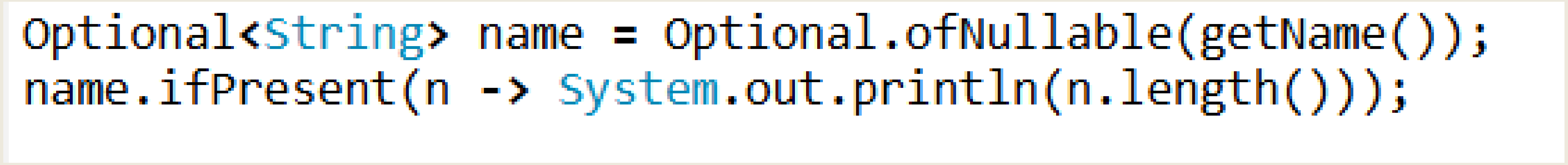
Helps handle null safely.

Before Java 8:



```java
String name = getName(); // Might return null
if (name != null) {
    System.out.println(name.length());
}
```

With Java 8:

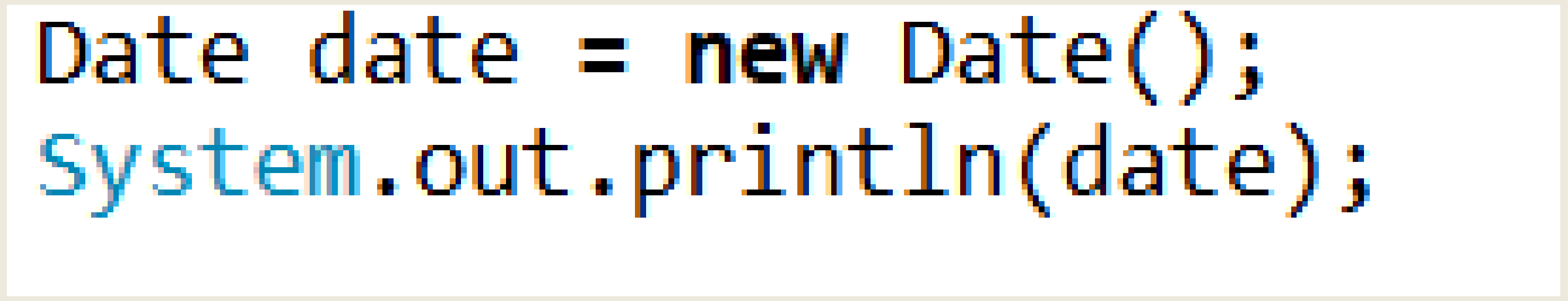


```java
Optional<String> name = Optional.ofNullable(getName());
name.ifPresent(n -> System.out.println(n.length()));
```

7. DateTime API (java.time)

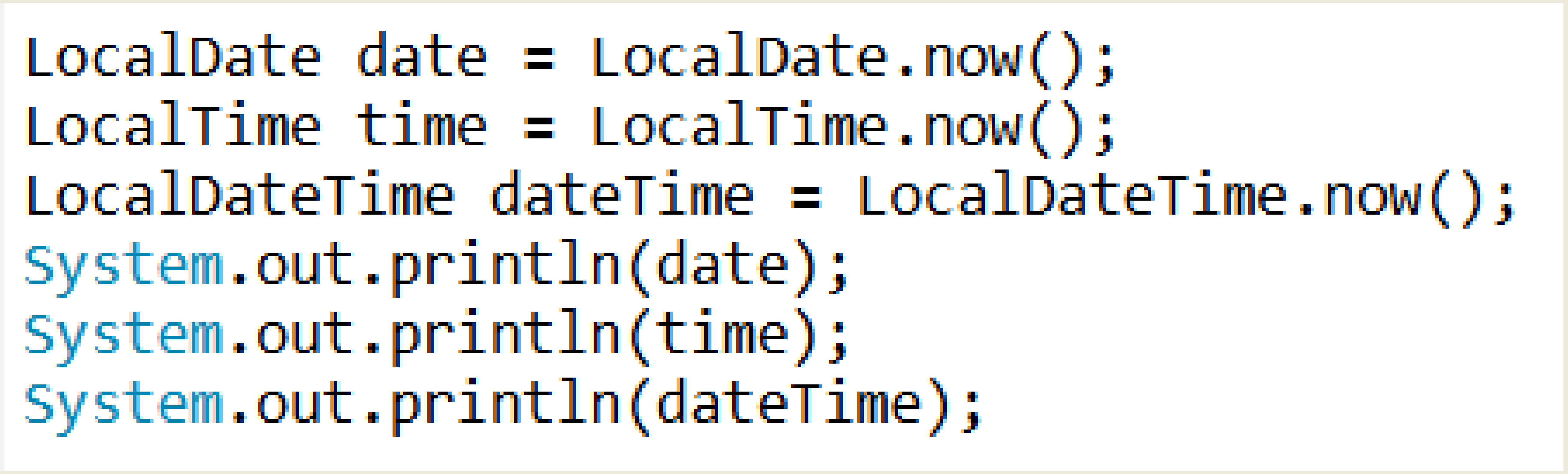
The old Date and Calendar APIs were mutable and error-prone.

Before Java 8:



```java
Date date = new Date();
System.out.println(date);
```

With Java 8:



```java
LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
LocalDateTime dateTime = LocalDateTime.now();
System.out.println(date);
System.out.println(time);
System.out.println(dateTime);
```

Java 8 wasn't just a version update — it changed how we think in Java. Whether it’s writing clean code, reducing boilerplate, or embracing functional programming, Java 8 has set a strong foundation.

💬 **Are you still writing Java the old way? It's time to level up!** 💻🔥