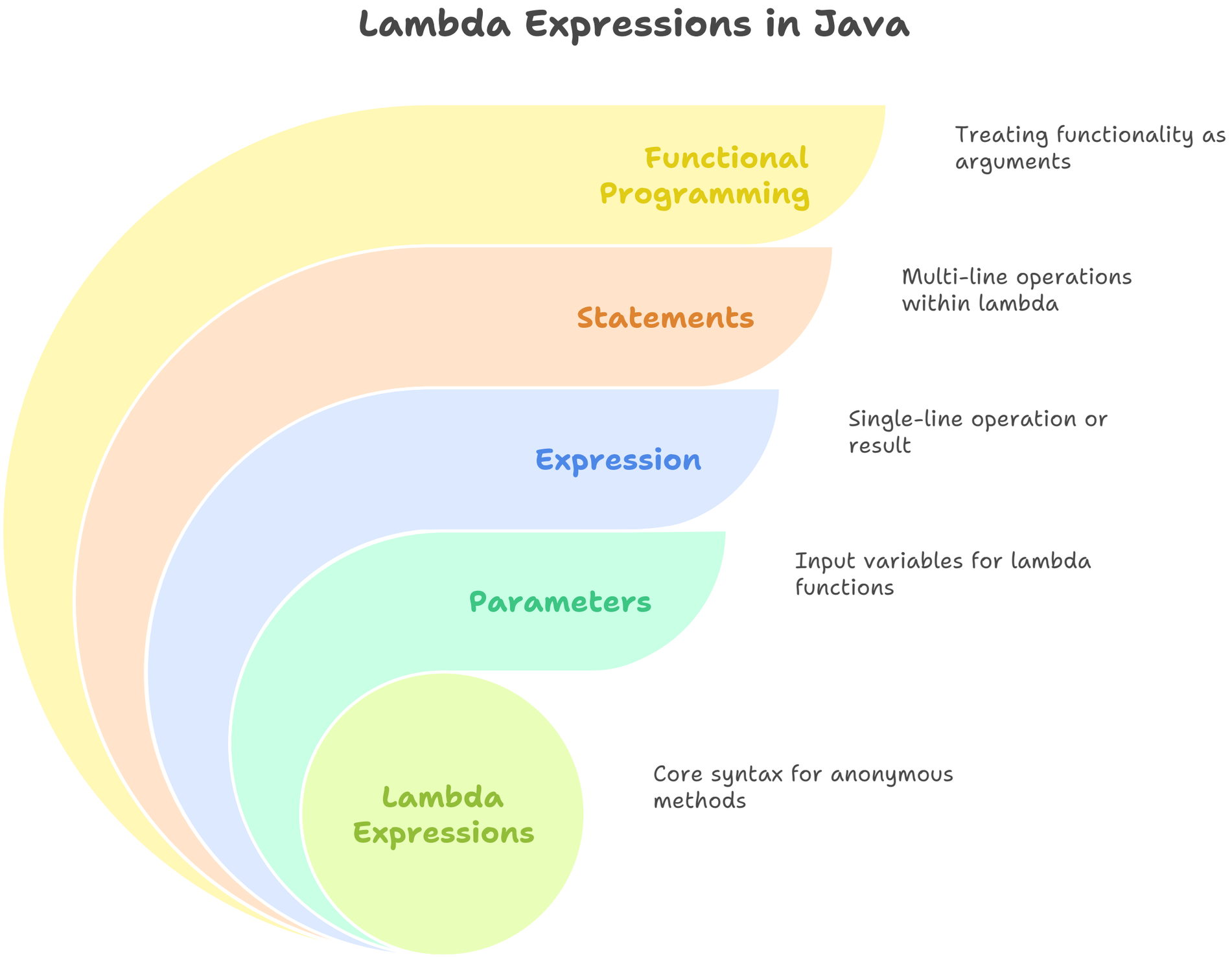
Java 8 Features

Java 8 introduced a significant number of enhancements and new features that transformed the way developers write Java code. This document outlines the key features of Java 8, highlighting their importance and how they improve the overall programming experience. From lambda expressions to the Stream API, these features not only simplify coding but also enhance performance and readability.

1. Lambda Expressions

Lambda expressions are a way to provide clear and concise syntax for writing anonymous methods. They enable you to treat functionality as a method argument, which is a fundamental aspect of functional programming. The syntax is (parameters) -> expression or (parameters) -> { statements; }.

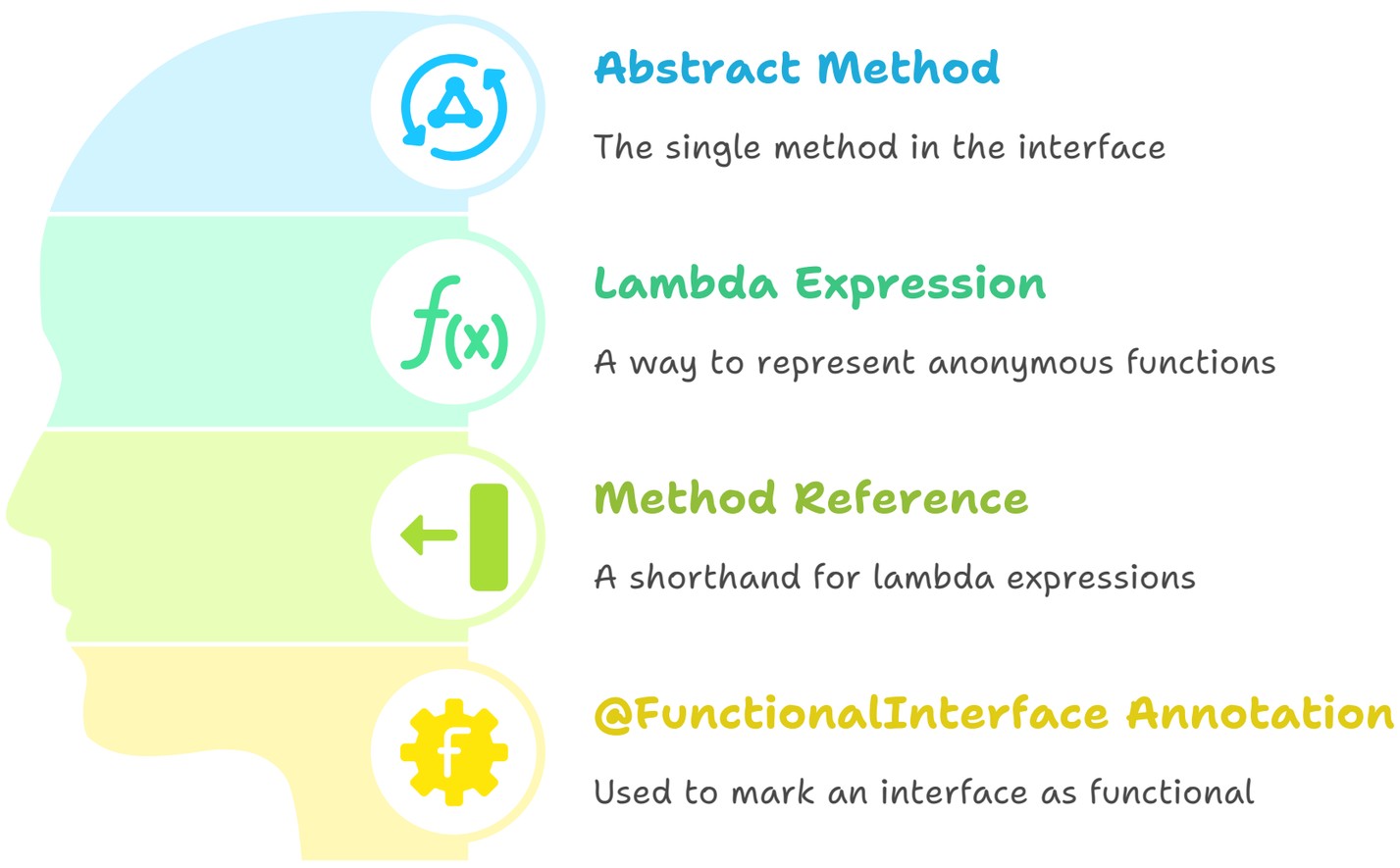


Example:

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.forEach(name -> System.out.println(name));
```

2. Functional Interfaces

A functional interface is an interface that contains exactly one abstract method. They can be used as the assignment target for a lambda expression or method reference. The @FunctionalInterface annotation is used to indicate that an interface is intended to be a functional interface.

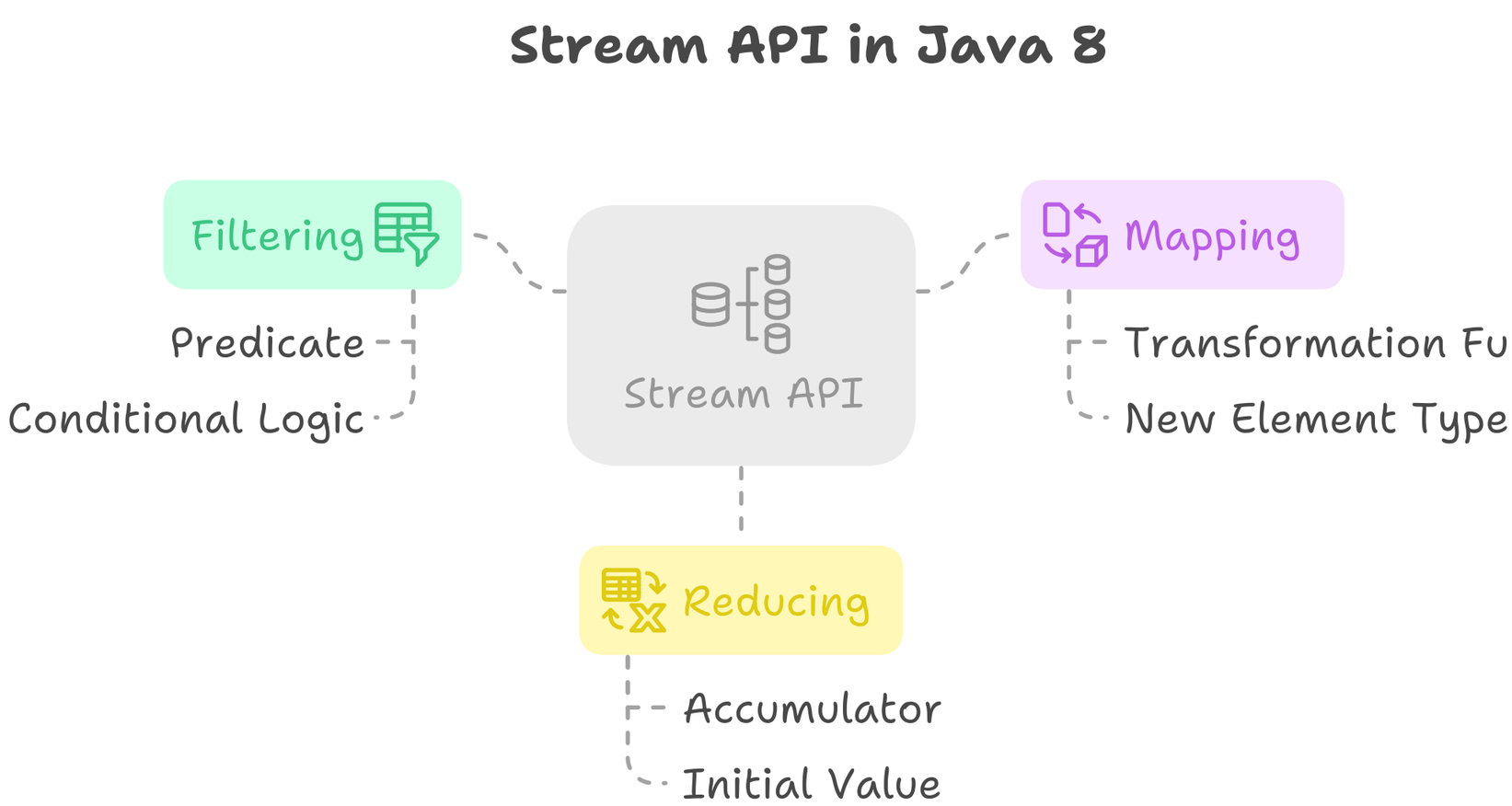


Example:

```
@FunctionalInterface
interface MyFunctionalInterface {
    void execute();
}
```

3. Stream API

The Stream API allows for functional-style operations on collections of objects. It provides a high-level abstraction for processing sequences of elements, enabling operations like filtering, mapping, and reducing.



Example:

```
List<String> filteredNames = names.stream()
                                  .filter(name -> name.startsWith("A"))
                                  .collect(Collectors.toList());
```

4. Default Methods

Java 8 allows interfaces to have default methods with an implementation. This feature enables developers to add new methods to interfaces without breaking existing implementations.

Example:

```
interface MyInterface {
    default void defaultMethod() {
        System.out.println("Default Method");
    }
}
```

5. Method References

Method references provide a way to refer to methods without invoking them. They can be used to simplify lambda expressions and improve code readability.

Example:

```
names.forEach(System.out::println);
```

6. Optional Class

The **Optional** class is a container object which may or may not contain a value. It is used to avoid null checks and **NullPointerExceptions**, providing a more expressive way to handle optional values.

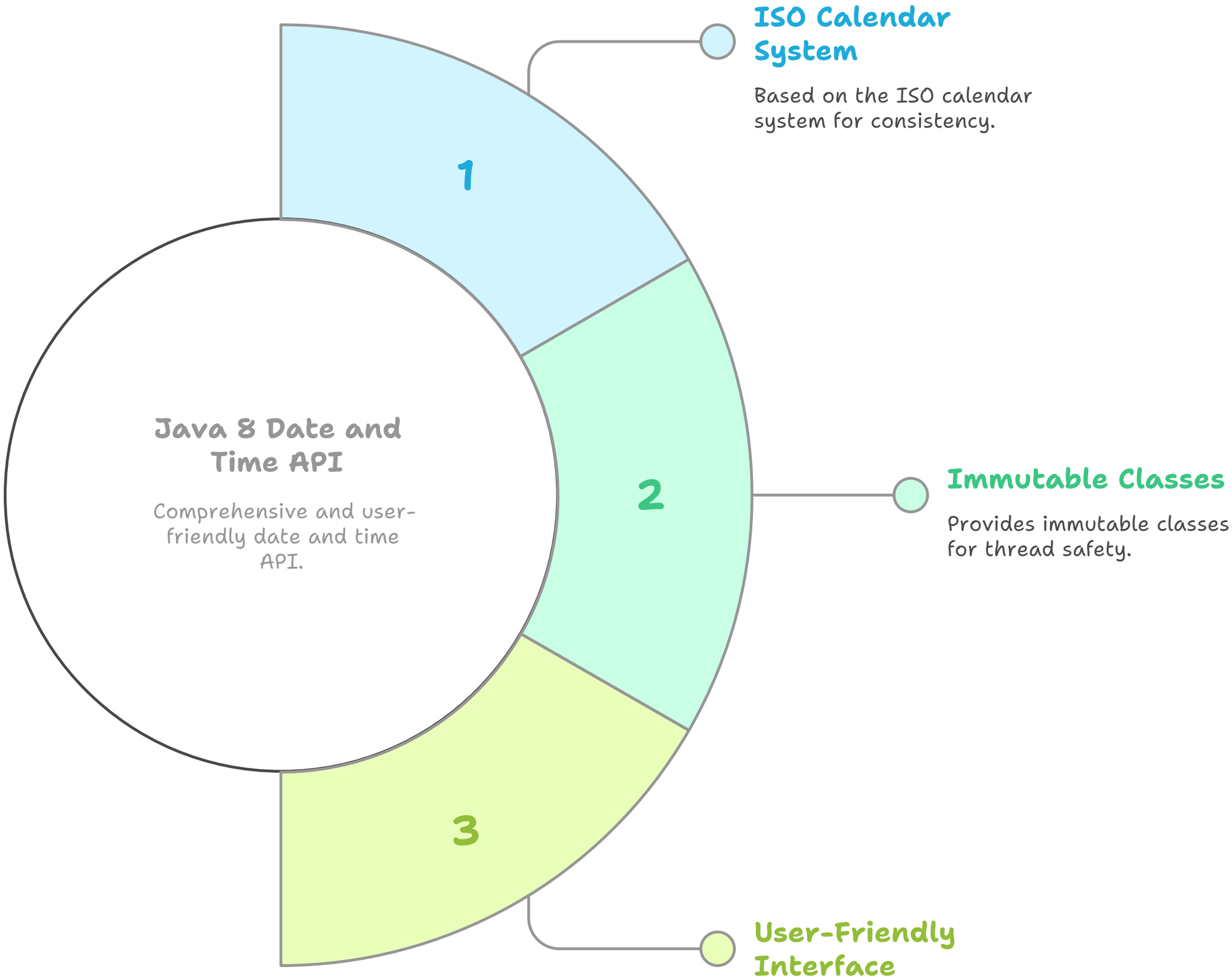
Example:

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

7. New Date and Time API

Java 8 introduced a new Date and Time API that is more comprehensive and user-friendly than the previous java.util.Date and java.util.Calendar. The new API is based on the ISO calendar system and provides immutable classes.

Exploring Java 8's Date and Time API



Example:

```
LocalDate today = LocalDate.now();
LocalDate nextWeek = today.plusWeeks(1);
```

8. Nashorn JavaScript Engine

Java 8 includes the Nashorn JavaScript engine, which allows developers to embed JavaScript code within Java applications. It provides a lightweight and high-performance way to execute JavaScript.

Example:

```
ScriptEngine engine = new ScriptEngineManager().getEngineByName("Nashorn");
engine.eval("print('Hello from JavaScript!');");
```

Conclusion

Java 8 has brought numerous features that enhance the language and make it more powerful and expressive. The introduction of lambda expressions, the Stream API, and the new Date and Time API are just a few examples of how Java 8 has evolved to meet modern programming needs. Embracing these features can lead to cleaner, more efficient, and more maintainable code.