

Overview of Java's Supported Programming Paradigms

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Lesson

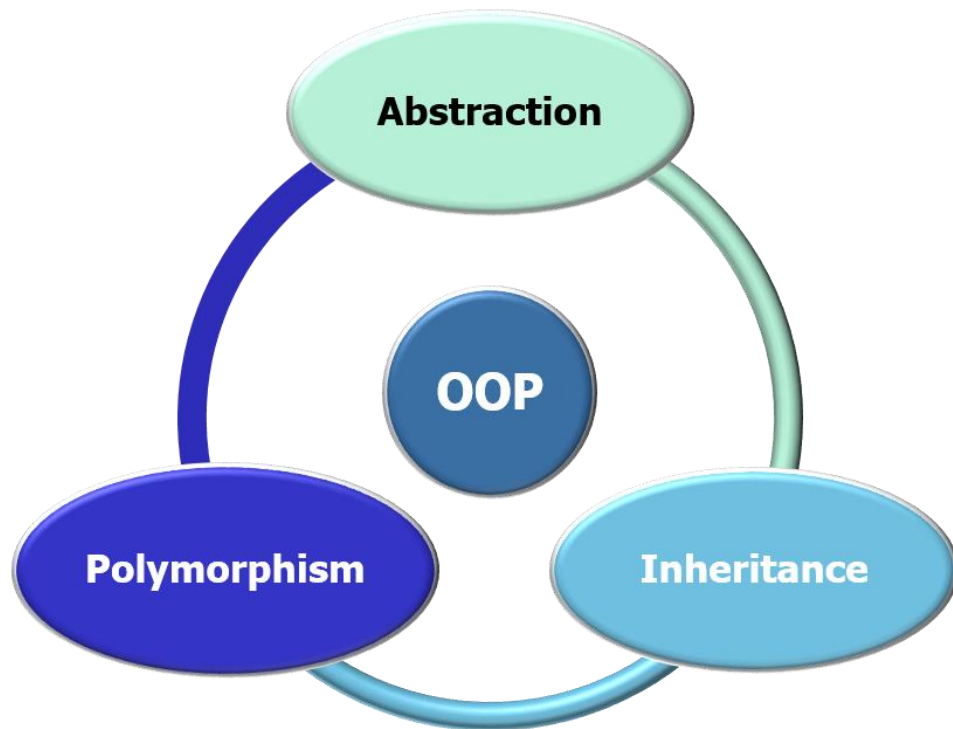
- Recognize the two programming paradigms supported by modern Java



Naturally, these paradigms are also supported in versions above & beyond Java 8!

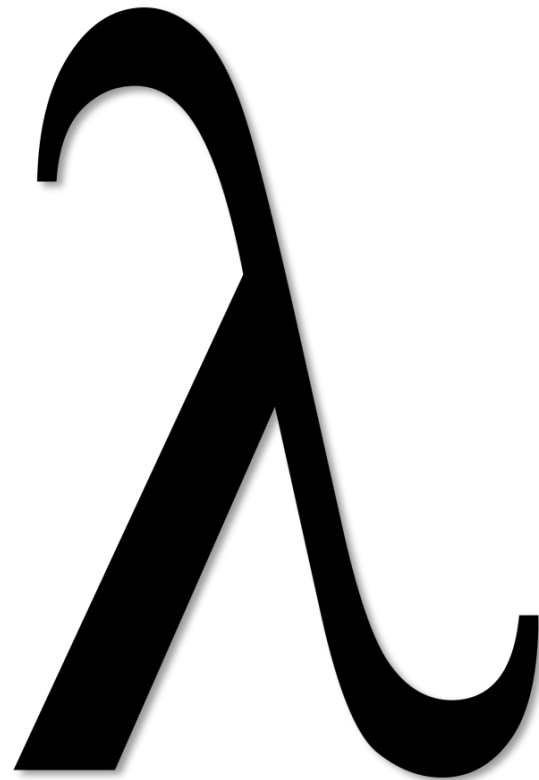
Learning Objectives in this Lesson

- Recognize the two programming paradigms supported by modern Java
 - Object-oriented programming



Learning Objectives in this Lesson

- Recognize the two programming paradigms supported by modern Java
 - Object-oriented programming
 - Functional programming



Learning Objectives in this Lesson

- Recognize the two programming paradigms supported by modern Java
 - Object-oriented programming
 - Functional programming

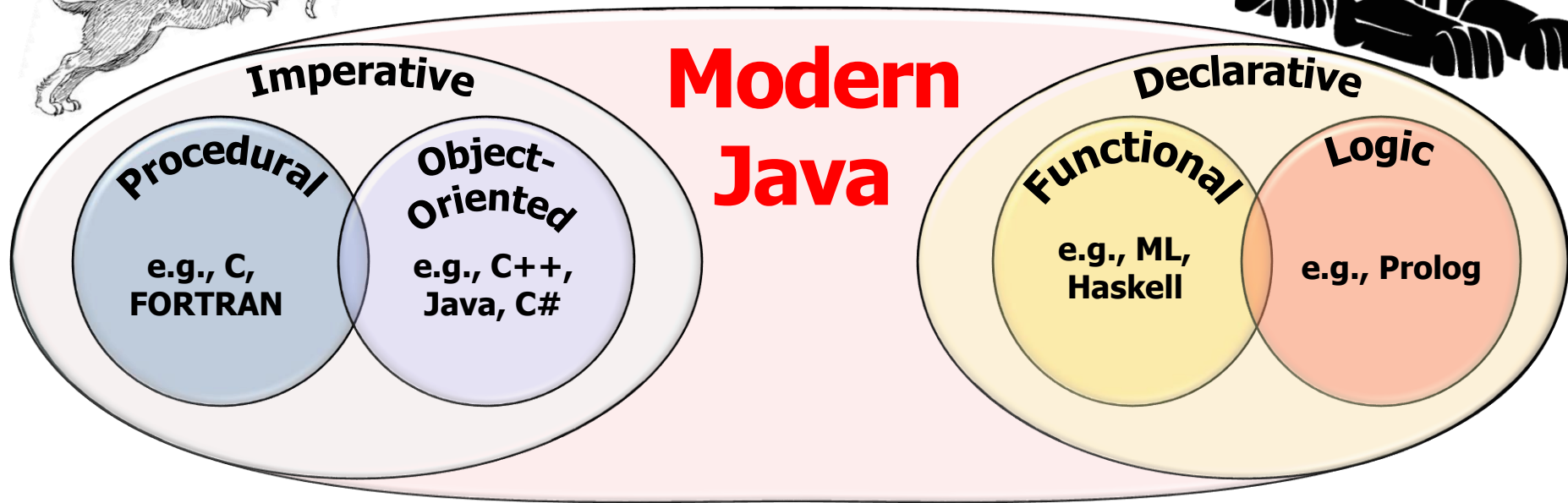


We show some modern Java code fragments that we'll cover in more detail later

Overview of Programming Paradigms in Modern Java

Overview of Programming Paradigms in Modern Java

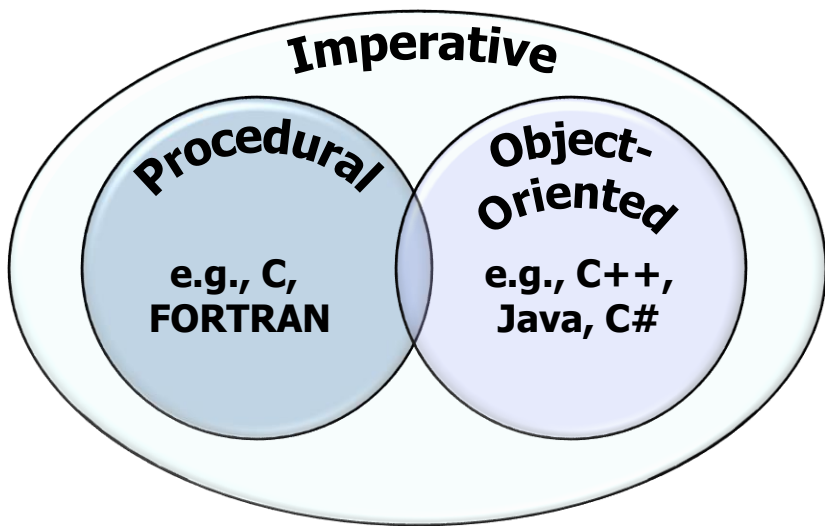
- Modern Java is a “hybrid” combining object-oriented & functional paradigms



See www.deadcoderising.com/why-you-should-embrace-lambdas-in-java-8

Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm

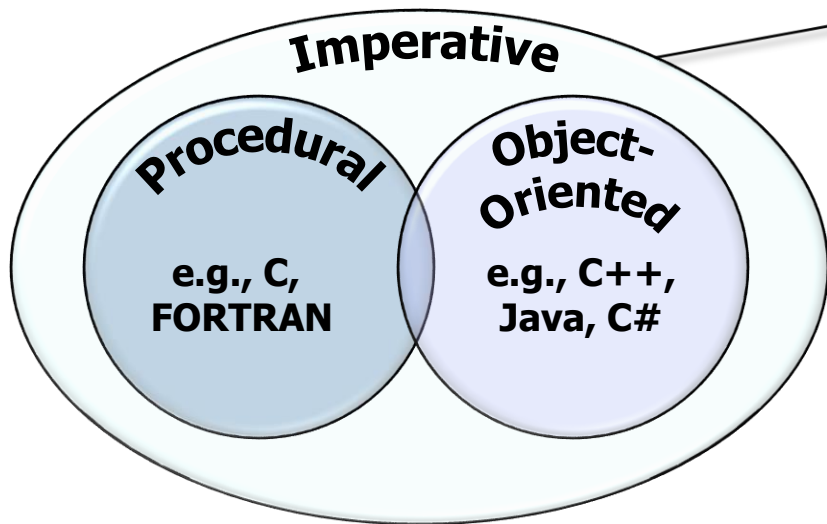


See en.wikipedia.org/wiki/Imperative_programming

Overview of Programming Paradigms in Modern Java

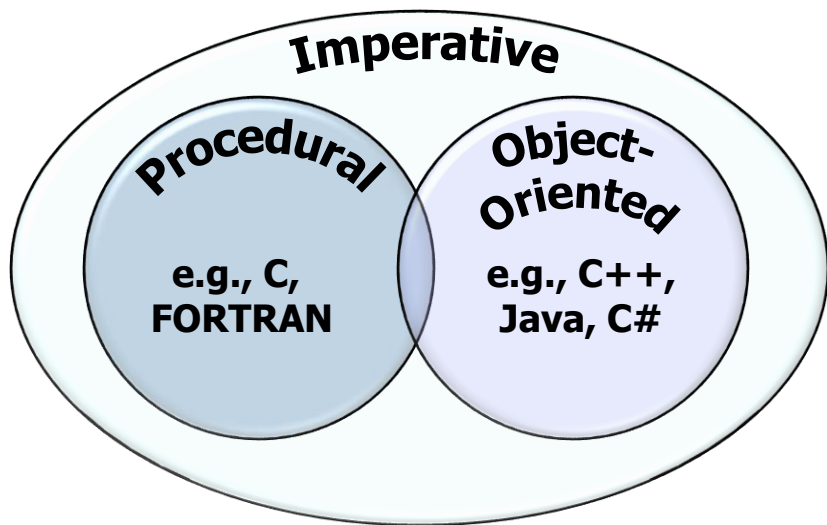
- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform

Imperative programming focuses on describing how a program operates via statements that change its state



Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform



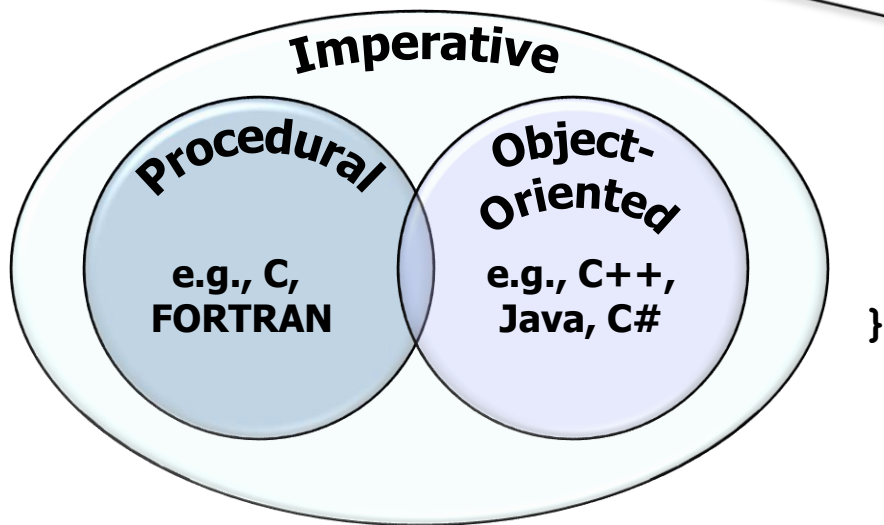
```
List<String> zap(List<String> lines,  
                String omit) {  
    List<String> res =  
        new ArrayList<>();  
    for (String line : lines)  
        if (!omit.equals(line))  
            res.add(line);  
    return res;  
}
```

Imperatively remove a given string from a list of strings

Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform

Create an empty list to hold results



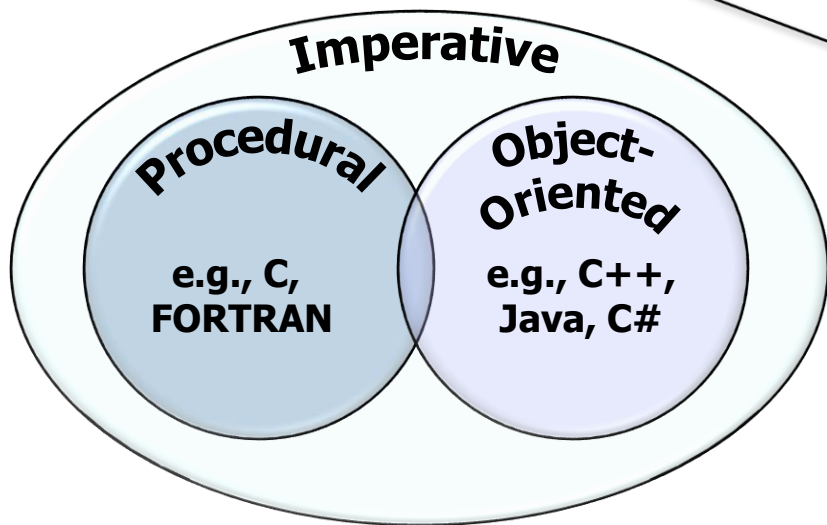
```
List<String> zap(List<String> lines,  
                String omit) {  
    List<String> res =  
        new ArrayList<>();  
    for (String line : lines)  
        if (!omit.equals(line))  
            res.add(line);  
    return res;  
}
```

Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform

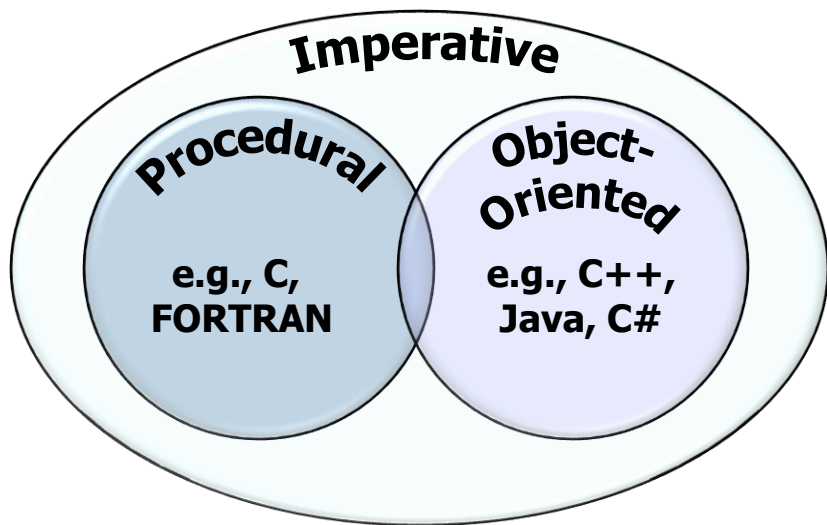
```
List<String> zap(List<String> lines,  
                String omit) {  
    List<String> res =  
        new ArrayList<>();  
    for (String line : lines)  
        if (!omit.equals(line))  
            res.add(line);  
    return res;  
}
```

Iterate sequentially through each line



Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform

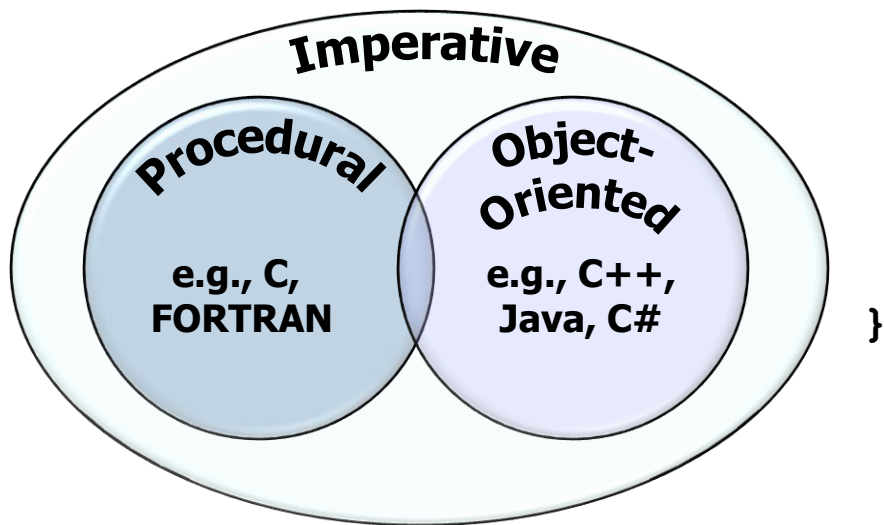


```
List<String> zap(List<String> lines,  
                String omit) {  
    List<String> res =  
        new ArrayList<>();  
    for (String line : lines)  
        if (!omit.equals(line))  
            res.add(line);  
    return res;  
}
```

*Only add lines that don't
match the omit string*

Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform



```
List<String> zap(List<String> lines,  
                String omit) {  
    List<String> res =  
        new ArrayList<>();  
    for (String line : lines)  
        if (!omit.equals(line))  
            res.add(line);  
    return res;  
}
```

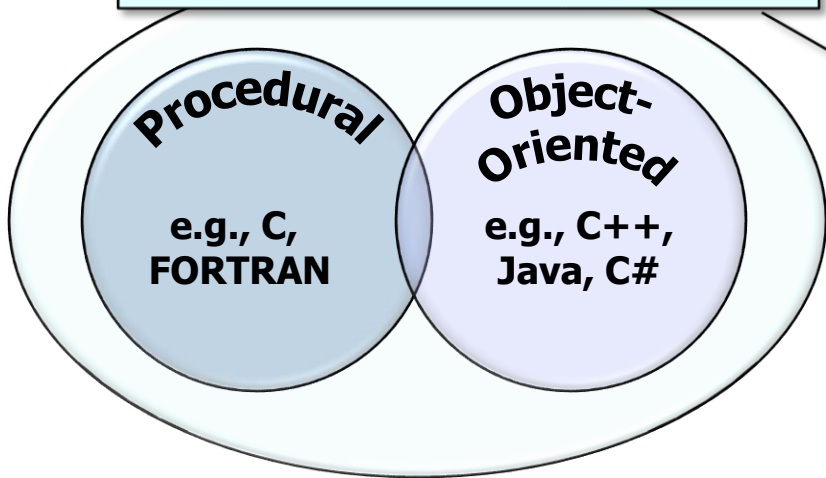
Return the list of non-matching lines

Overview of Programming Paradigms in Modern Java

- Object-oriented programming is an “imperative” paradigm
 - e.g., a program consists of commands for the computer to perform

```
List<String> zap(List<String> lines,  
                String omit) {  
    List<String> res =  
        new ArrayList<>();  
    for (String line : lines)  
        if (!omit.equals(line))  
            res.add(line);  
    return res;  
}
```

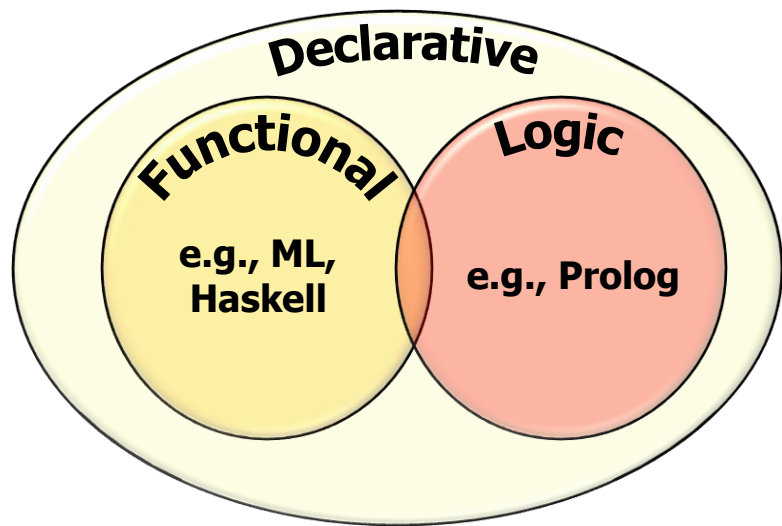
*This sequential code applies
the Accumulator anti-pattern*



See developer.ibm.com/articles/j-java-streams-2-brian-goetz

Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm

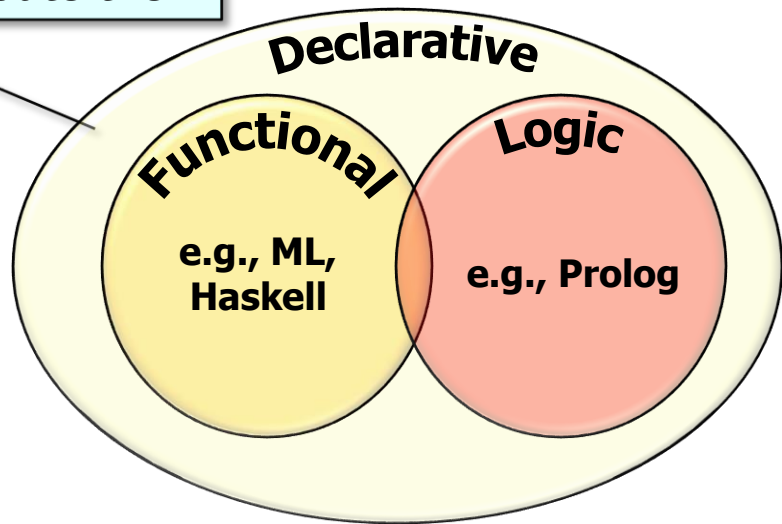
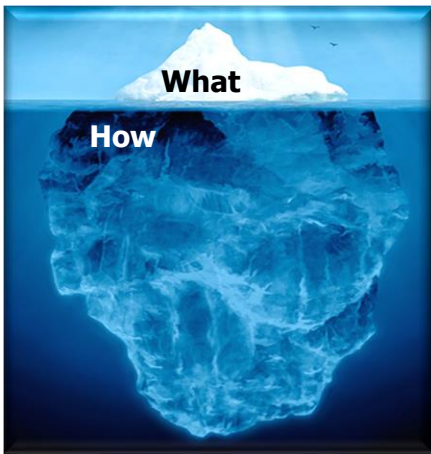


See en.wikipedia.org/wiki/Declarative_programming

Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

Declarative programming focuses on “what” computations should be performed, instead of on “how” to compute them

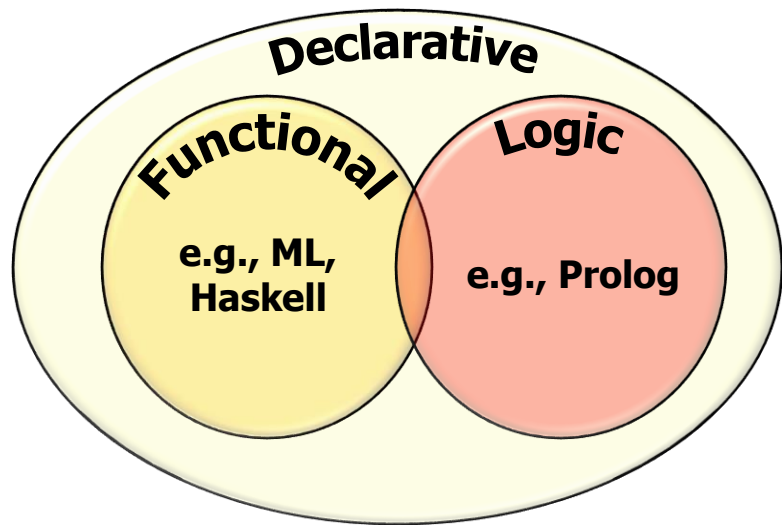


Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .stream()  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```

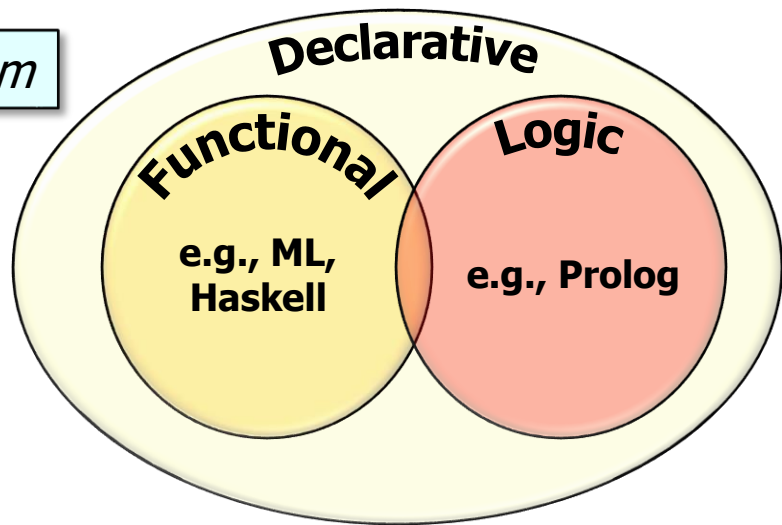
Declaratively remove a given string from a list of strings



Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .stream() ————— Convert list into a stream  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```



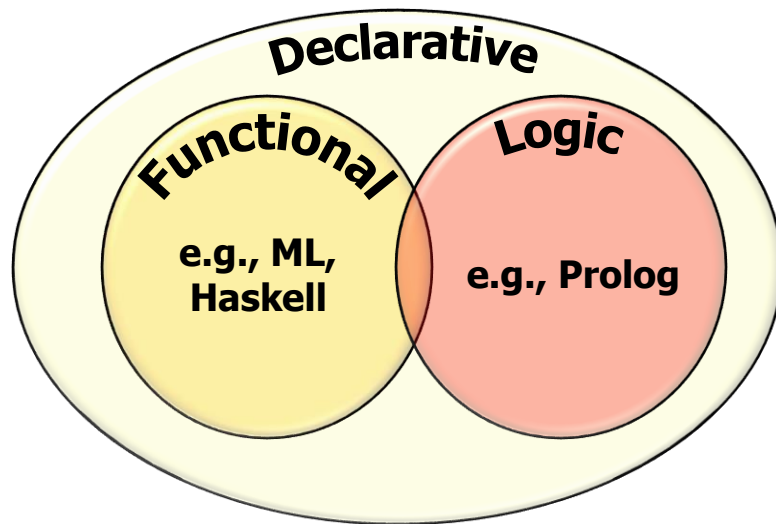
See docs.oracle.com/javase/tutorial/collections/streams

Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .stream()  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```

*Remove any line in the stream
that matches the omit param*

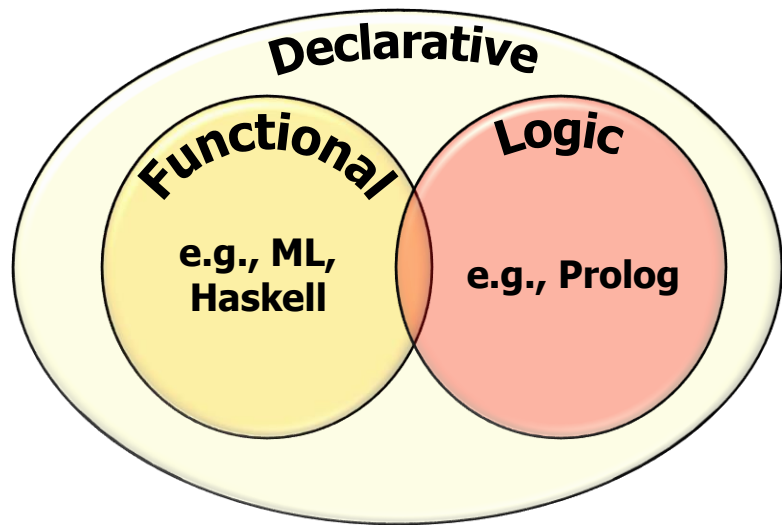


Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .stream()  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```

*Collect all non-matching lines
into a list & return it the caller*

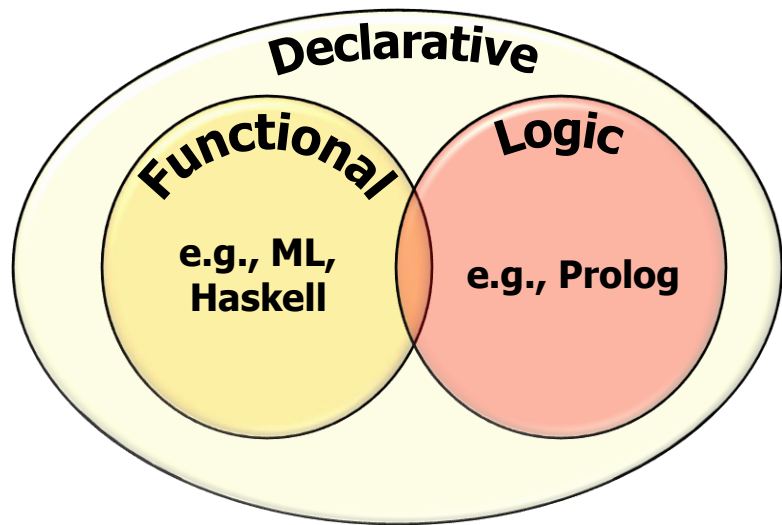


Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .stream()  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```

*Note “fluent” programming style
with cascading method calls*

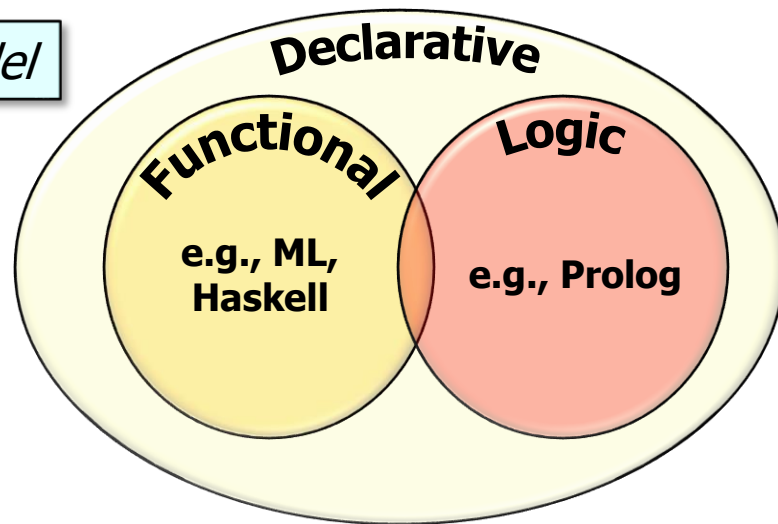


See en.wikipedia.org/wiki/Fluent_interface

Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .parallelStream() Filter in parallel  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```

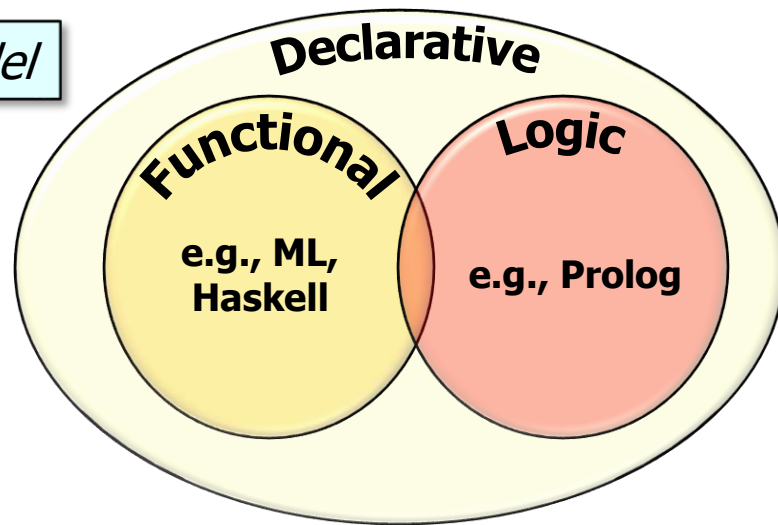
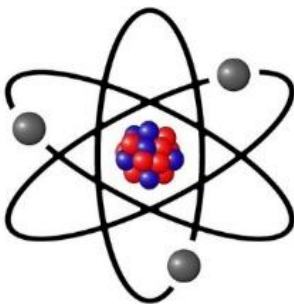


See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

Overview of Programming Paradigms in Modern Java

- Conversely, functional programming is a “declarative” paradigm
 - e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps

```
List<String> zap(List<String> lines,  
                String omit) {  
    return lines  
        .parallelStream() Filter in parallel  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```



Code was parallelized with minuscule changes since it's declarative & stateless!

Overview of Programming Paradigms in Modern Java

- Summary of these two paradigms



The image is a screenshot of a Twitter interface. At the top, there is a search bar with the Twitter logo and the text "Search Twitter". To the right are buttons for "Log in" and "Sign up", along with a menu icon. The main content area features a tweet from Michael Feathers (@mfeathers) dated Nov 3, 2010. The tweet text is "OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts." and is highlighted with a red rectangular border. Below the tweet are icons for replies (8), retweets (457), likes (438), and a share icon. A "Replies" section follows, showing a reply from Kenji Hiranabe (@hiranabe) dated Jul 22, 2014. The reply text is in Japanese and mentions "@mfeathers". Below the reply are icons for replies (1), retweets (3), likes (2), and a share icon. On the right side of the interface, there is a "New to Twitter?" section with a "Sign up" button, and a "Relevant people" section featuring Michael Feathers with a "Follow" button.

Michael Feathers @mfeathers · Nov 3, 2010

OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.

8 457 438

Replies

Kenji Hiranabe @hiranabe · Jul 22, 2014

Replying to @mfeathers

オブジェクト指向が変化をカプセル化してコードの理解性を上げたのに対し、関数型は変化を最小化してコードの理解性を上げた。TRT "@mfeathers: OO makes code understandable by encapsulating moving..."

1 3 2

New to Twitter?

Sign up now to get your own personalized timeline!

Sign up

Relevant people

Michael Feathers @mfeathers

Director, R7K Research & Conveyance.
Author of Working Effectively with Legacy Code.

Follow

See twitter.com/mfeathers/status/29581296216?lang=en

Overview of Programming Paradigms in Modern Java

- Summary of these two paradigms:
 - Java's object-oriented programming features make code understandable by encapsulating the moving parts

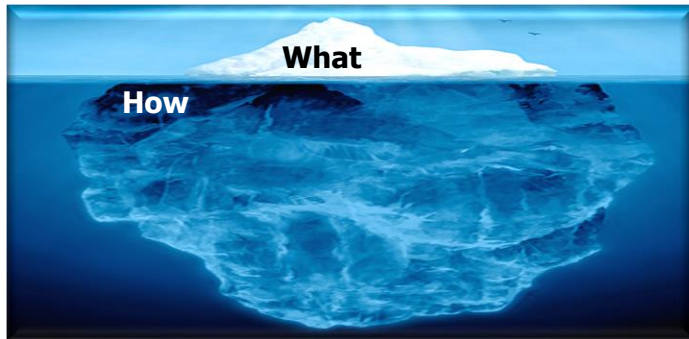


```
List<String> zap
(List<String> lines,
 String omit) {
    List<String> res =
        new ArrayList<>();
    for (String line : lines)
        if (!omit.equals(line))
            res.add(line);
    return res;
}
```

Overview of Programming Paradigms in Modern Java

- Summary of these two paradigms:
 - Java's object-oriented programming features make code understandable by encapsulating the moving parts
 - It's functional programming features make code understandable by eliminating the moving parts

```
List<String> zap  
    (List<String> lines,  
     String omit) {  
    return lines  
        .parallelStream()  
        .filter(not(omit::equals))  
        .collect(toList());  
}
```



End of Overview of Java's Supported Programming Paradigms