The difference between executing run and start on a thread

There's a big difference between calling `run()` and `start()`.

If you execute the **run** method, it's executed **synchronously**, by the running thread it's invoked from.

If you want your code to be run **asynchronously**, **you must call the start method**.

The native modifier on a method

The `native` modifier indicates that this method's source code isn't written in Java.

It's written in another language, such as C or C++.

The code in this example, is part of a native library, such as a dll file.

Why use a native library?

The reasons to do this include

- To access system-level functionality that's platform-specific.
- To interface with hardware.
- To optimize performance for tasks that might be computationally-intensive.

Runnable is a Functional Interface

It's important to recognize that Runnable is a functional interface.

It's functional method, or its single access method, is the run method.

Anywhere you see a Runnable type, it's a target for a lambda expression.

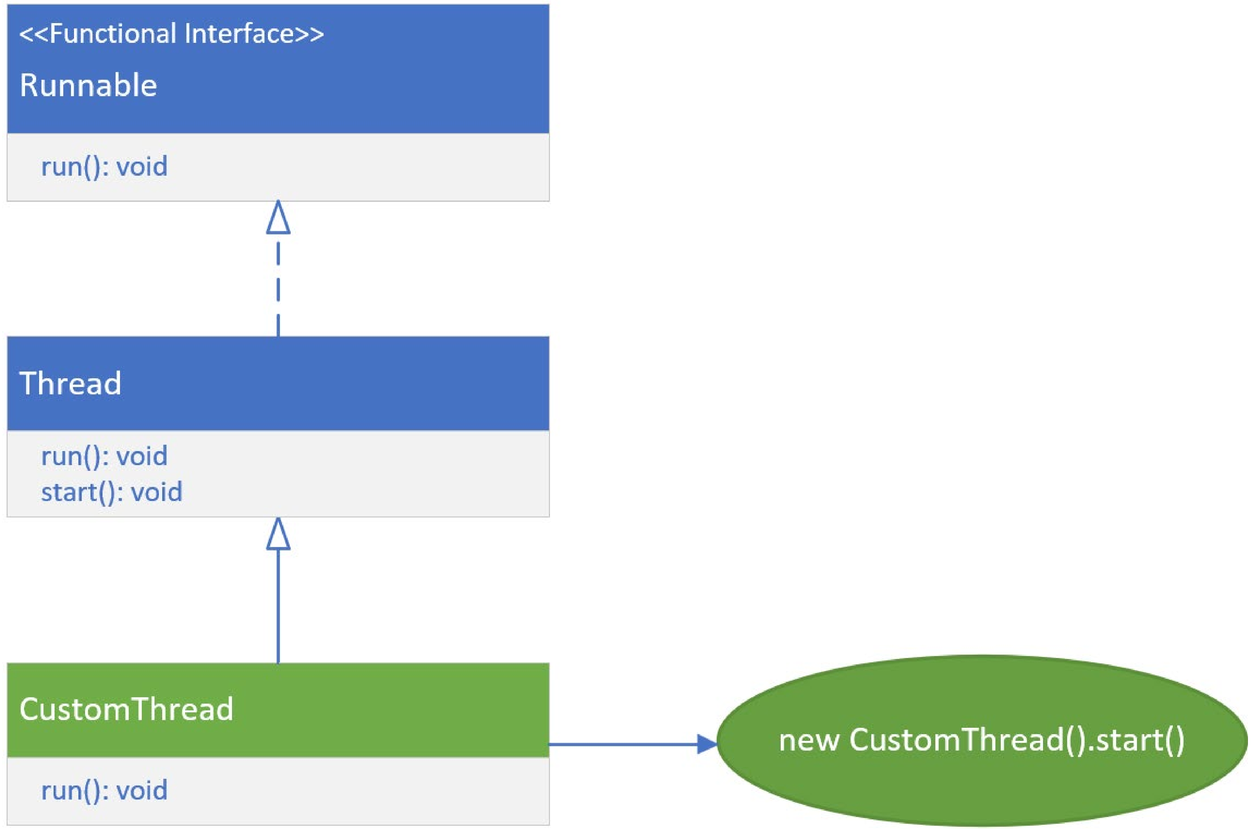
You can have any class implement the Runnable interface to run asynchronously.

Extending the Thread class

This slide demonstrates extending the Thread class.

The new subclass overrides the Thread's run method, to provide the concurrent thread's task.

To use this thread, you create a new instance of your subclass, with a no argument constructor, and execute the start method on that instance.



Advantages of Extending Thread

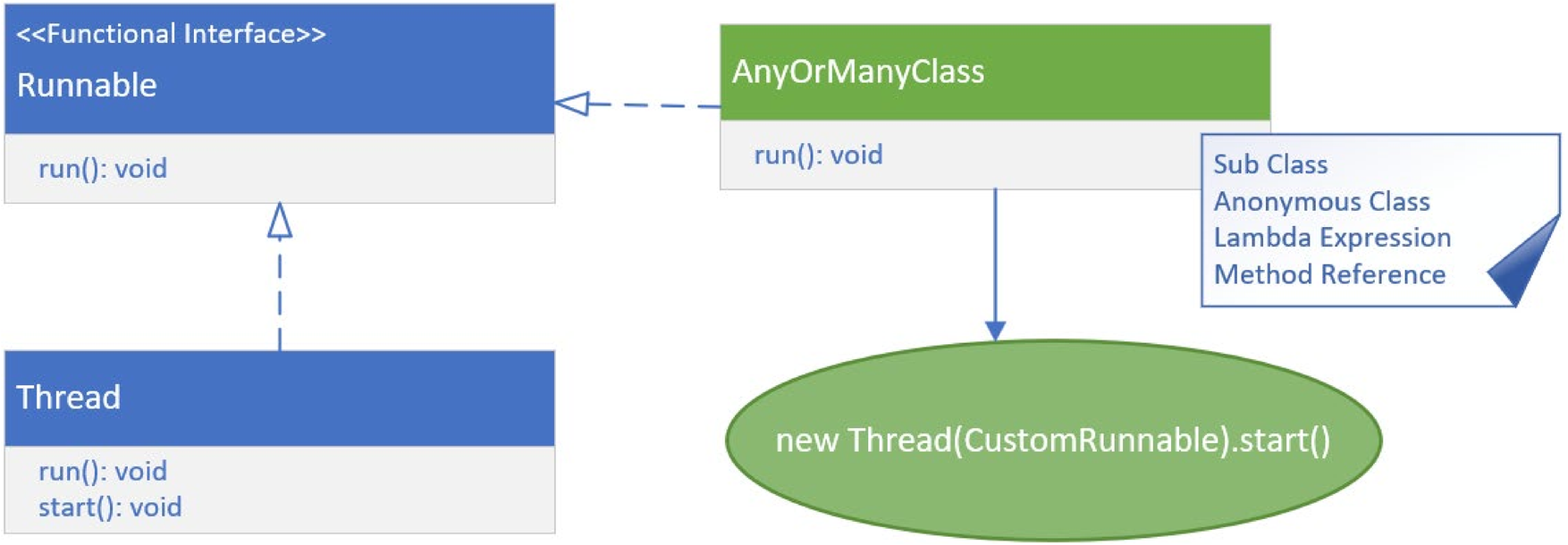
- You have more control over the thread's behavior and properties.
- You can access the thread's methods and fields directly from your sub class.
- You can create a new thread for each task.

Disadvantages of Extending Thread

You can only extend one class in Java, so your subclass can't extend any other classes.

Your class is tightly coupled to the Thread class, which may make it difficult to maintain.

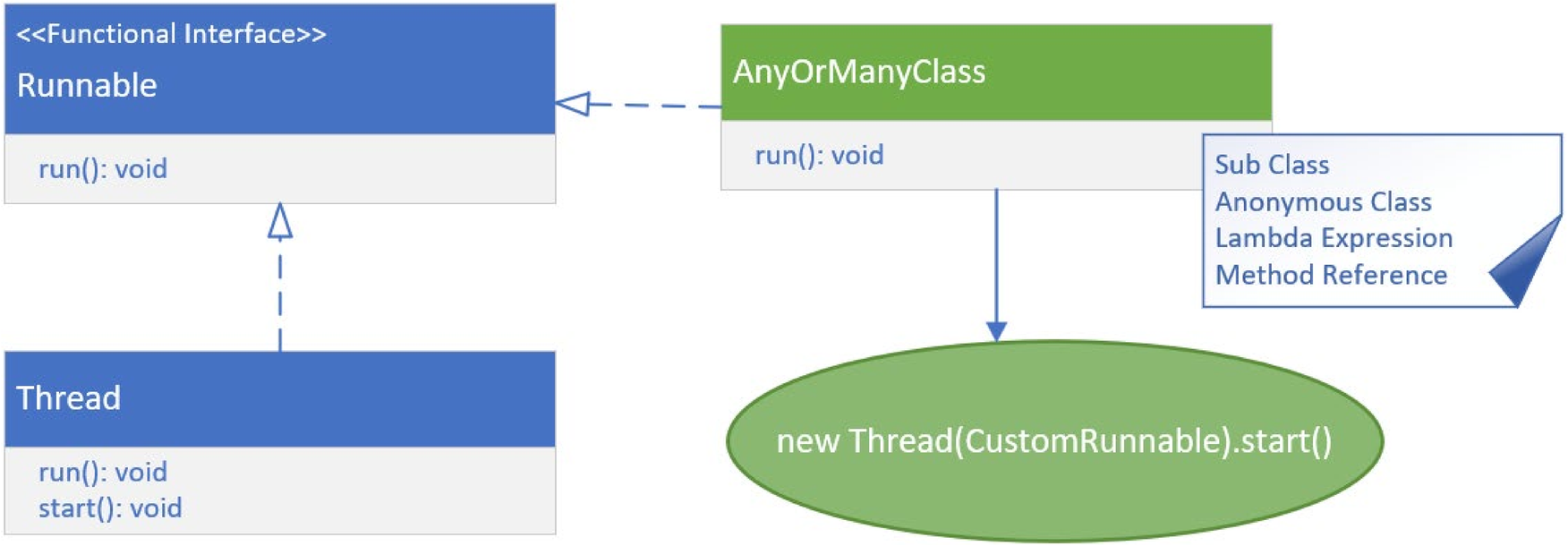
Implementing Runnable



Besides extending the Thread class, you can create threads, by implementing the Runnable interface.

This method allows any class, to implement Runnable, meaning any class at all, can be used in a thread.

Implementing Runnable



This class is passed to the Thread constructor, that accepts a Runnable.

You can also pass an anonymous class, lambda expression, or a method reference to this constructor, to create an instance of a Thread.

You again call start on the new thread instance, to execute the code asynchronously.

Advantages of Implementing a Runnable and creating a Thread instance with it

- You can extend any class and still implement Runnable.
- Your class (if you create a class) is loosely coupled to the Thread class, which makes it easier to maintain.
- You can use anonymous classes, lambda expressions, or method references, to very quickly describe thread behavior.

Disadvantages of Implementing a Runnable and creating a Thread instance with it

- You do have less control over the thread's behavior and properties.

In other words, You can't access the thread's methods and fields directly, from the run method.