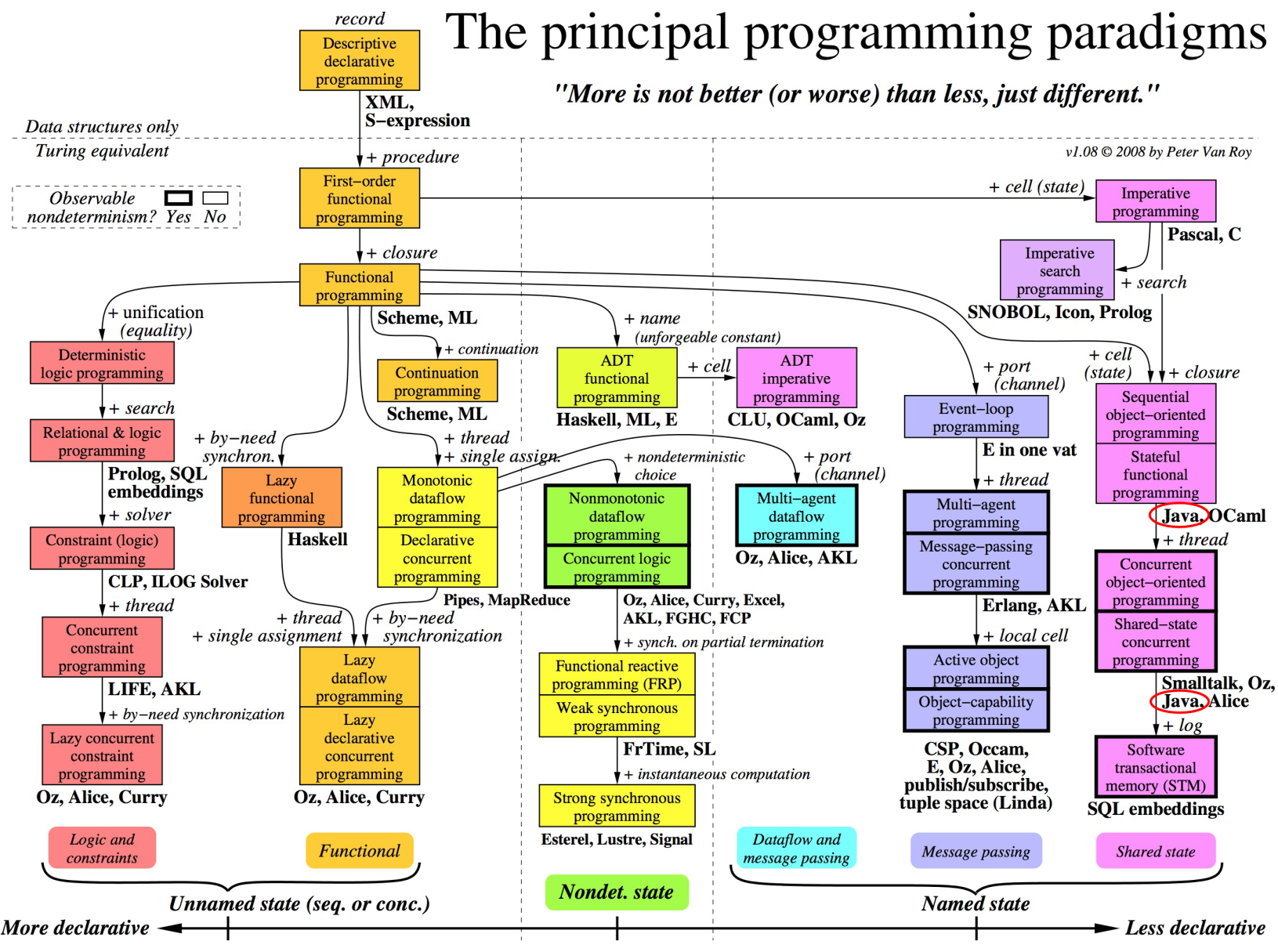
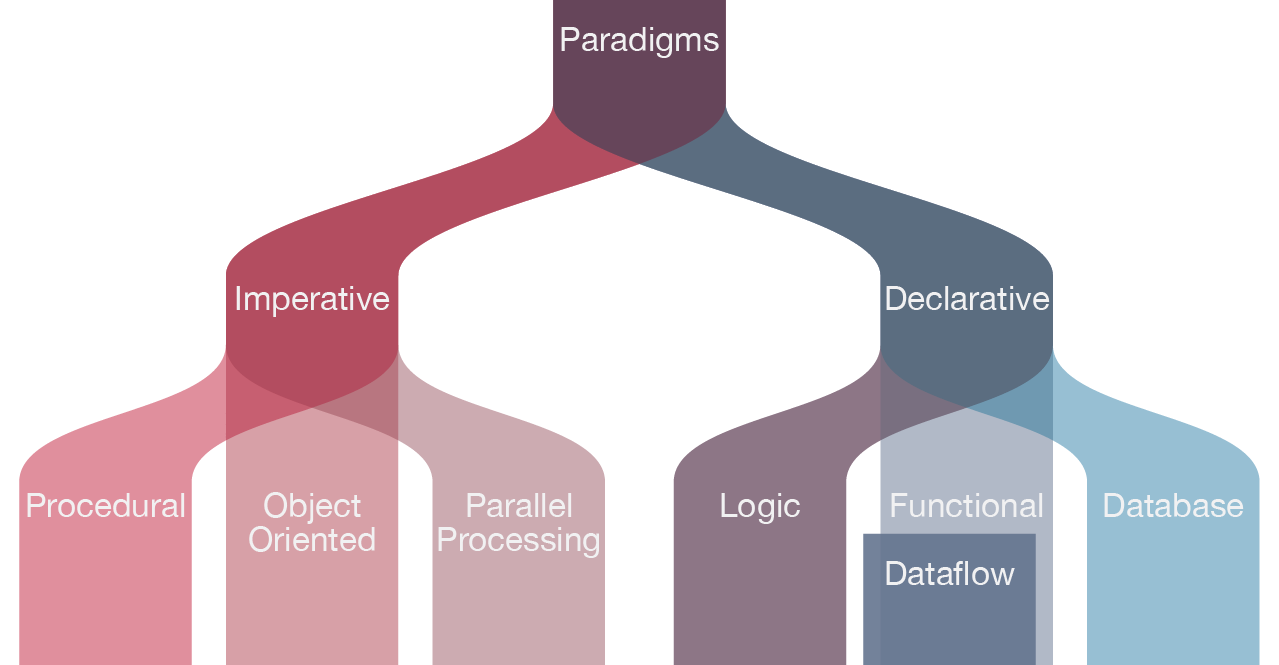
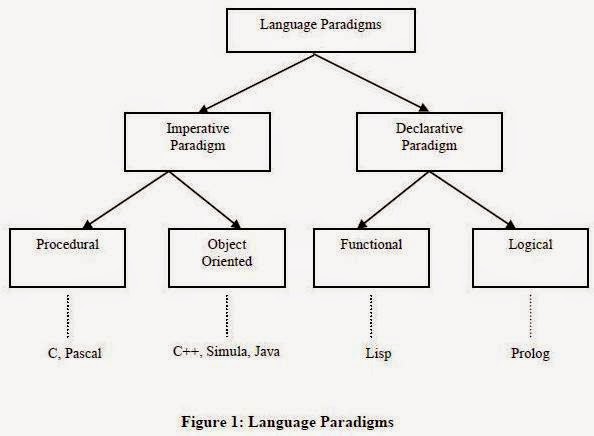
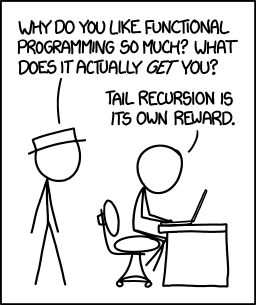
****

รูป 0‑1 The principal programming paradigm

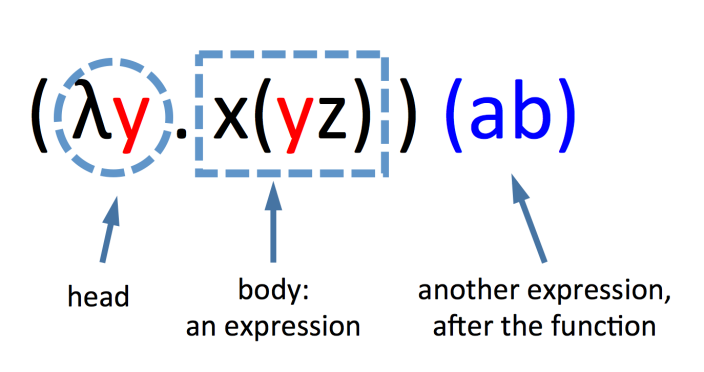
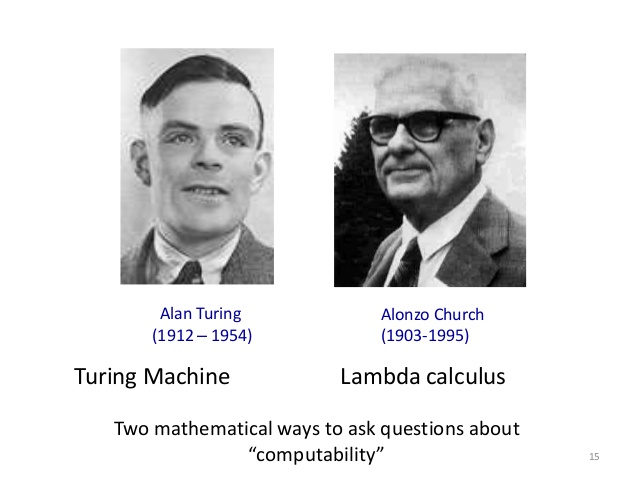


รูป 0‑2 ประเภทของภาษา

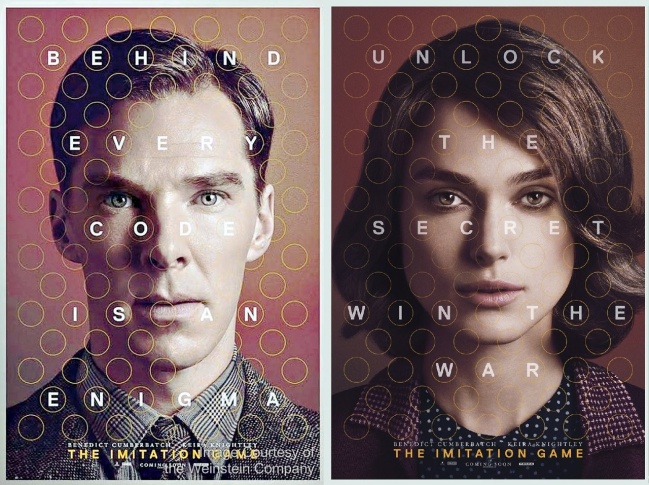


 ****

**Functional Programming**



มีพื้นฐานมาจาก Lambda calculus ซึ่งเป็นคณิตศาสตร์แขนงหนึ่ง ซึ่งคิดค้นโดย Alonzo Church ผู้เป็นอาจารย์ของ Alan Turing เจ้าของ Turing machine (คนเดียวกับตัวละครในภาพยนตร์เรื่อง The Imitation Game) เครื่องที่เป็นแบบจำลองทางคณิตศาสตร์ที่มีความสามารถระดับเดียวกับคอมพิวเตอร์ในปัจจุบัน แต่มีหน่วยความจำไม่จำกัด



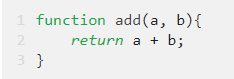
จากเครื่องนี้ เราจะสามารถอธิบาย Algorithm ใดๆก็ตามด้วย Turing machine ได้ ซึ่งภายหลัง Turing เองก็ได้ทำการพิสูจน์ว่า Lambda calculus กับ Turing machine นั้นสมมูลกัน สามารถใช้แทนกันได้ทุกกรณี นั่นคือ เราสามารถใช้ functional programming มา Implement algorithm ใดก็ได้ที่ computable หรือพูดง่ายๆก็คือ โปรแกรมที่เราเคยเขียนได้ สามารถแปลงมาเป็นแบบ Functional Programming ได้หมด

ตัวอย่างภาษาที่เห็นใช้ชัดเจน เช่น Lisp, Clojure, Scala, Erlang, Javascript ฯลฯ

**คุณสมบัติพื้นฐาน**

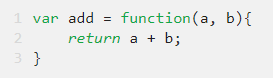
1. **Pure Function** **(มีคุณสมบัติ Idempotence)**

ฟังก์ชันรูปแบบที่ง่ายที่สุดฟังก์ชันแบบปกติ เรียกใช้งาน และได้ผลลัพธ์เท่าเดิม

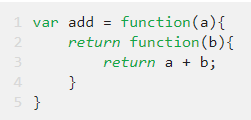
****

1. **First-Class Function (มีคุณสมบัติ Referential transparency)**

สามารถเอาฟังก์ชัน ใส่ตัวแปรได้ (ยกตัวอย่างเช่น ภาษา Javascript เป็นต้น) โดยจะเรียกว่า Anonymous Function



1. **Higher-Order Function**

ฟังก์ชั่นที่ Return ค่าคำตอบกลับมาเป็นฟังก์ชันได้ (ยกตัวอย่างเช่น ภาษา Javascript เป็นต้น) 

เวลาเรียกใช้งาน

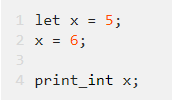


1. **Closures**

อ้างอิงจาก High-Order Function ตัว Closure คือ ตัวฟังก์ชันตัวในสามารถเรียกตัวแปรข้างนอก นำมาใช้ได้ทั้งหมด แต่ด้านนอกใช้ตัวแปรของข้างในไม่ได้

1. **Immutable State**

เก็บรักษา State เอาไว้ แม้ว่าจะมีการระบุค่าตัวใหม่ลงไปก็ตาม จะได้ค่าเท่าเดิม



**Functional Interfaces**

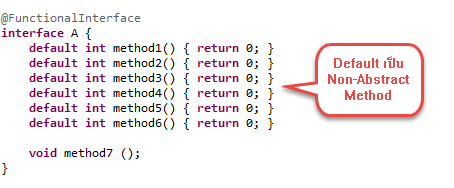
**จุดสำคัญ (Key Points)**

* A functional interface is any interface that has exactly one abstract method.

เป็น Interface ที่มีแค่ 1 Abstract Method เท่านั้น เรียกว่า **“Single Abstract Method (SAM) interfaces”**

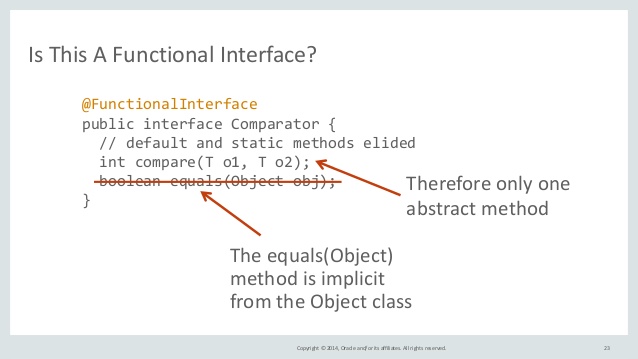
* Since default methods have an implementation, they are not abstract so a functional interface can have any number of them.

นับตั้งแต่นำ Default Methods มาใช้ และมันก็ไม่ได้เป็น Abstract ทำให้เราสามารถประกาศใช้เท่าไหร่ก็ได้

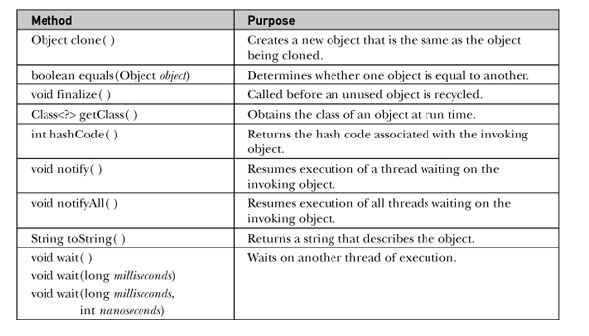


* If an interface declares an abstract method with the signature of one of the methods of java.lang.Object, it doesn't count toward the functional interface method count.

ถ้า Interface ประกาศ Abstract Method ที่เป็น Signature ของคลาส Object จะไม่ถูกนับเป็น Abstract Method ในเรื่อง Functional Interface



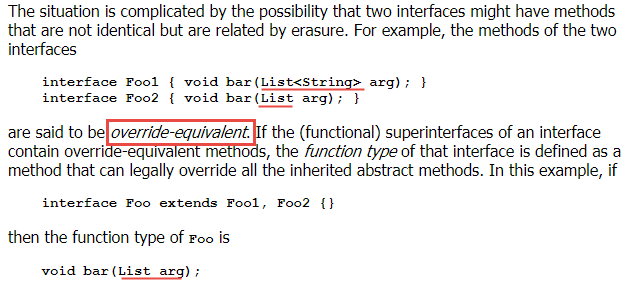
Signature of Class Object Method



**จุดสำคัญ (Key Points)** (ต่อ)

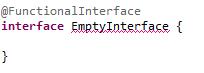
* A functional interface is valid when it inherits a method that is equivalent but not identical to another.

Functional Interface จะใช้ได้ก็ต่อเมื่อมัน inherit ตัว Method ที่คุณสมบัติเทียบเท่า แต่หน้าตาไม่เหมือนกัน



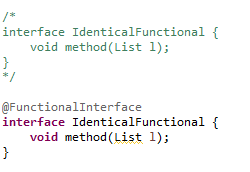
* An empty interface is not considered a functional interface.

Empty Interface จะไม่ถูกเรียกว่าเป็น Functional Interface

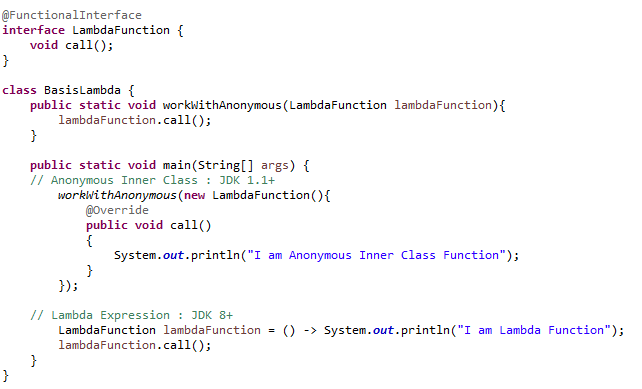


* A functional interface is valid even if the @FunctionalInterface annotation would be omitted.

Functional Interface สามารถใช้ได้ ถึงแม้จะละเว้นประกาศ @FunctionalInterface



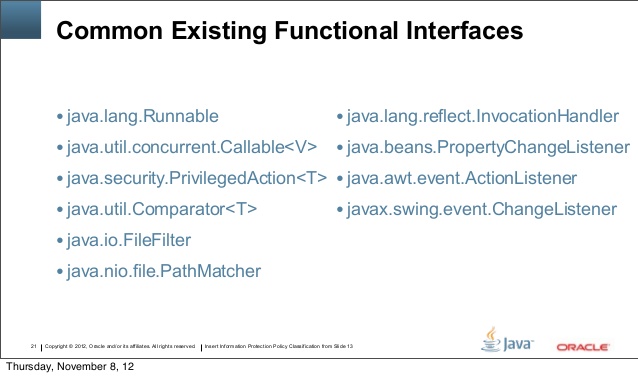
* Functional interfaces are the basis of lambda expressions.

Functional interface ทั้งหมดล้วนเป็นรากฐานของเรื่อง Lambda Expression (บทที่ 9)

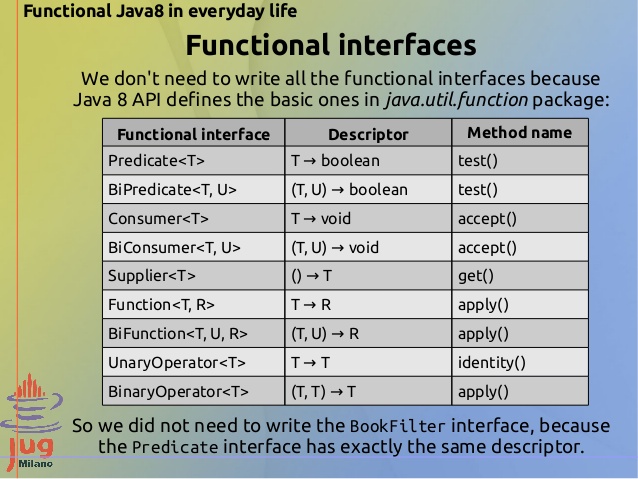
**บทสรุปการประกาศใช้ Functional Interface**



**Common Functional Interfaces in JDK 1-7**



**Newer in JDK 8**

****