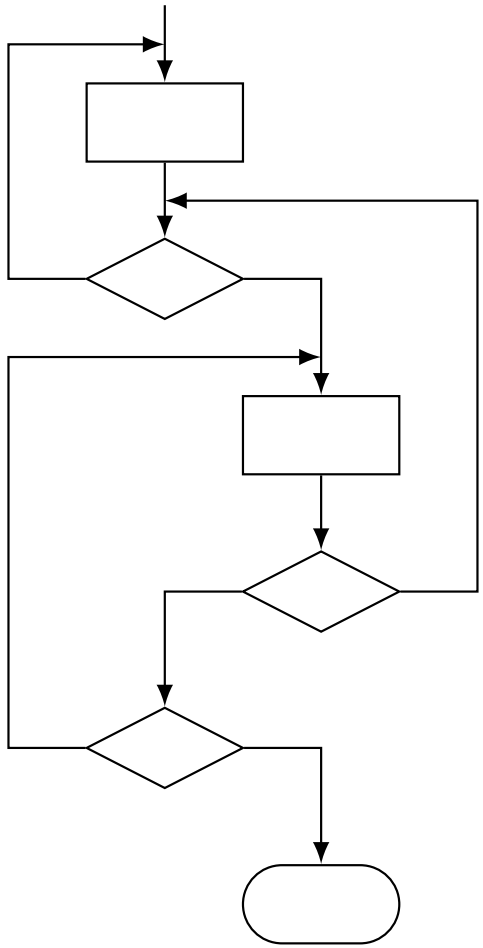
แนวคิด Object-Oriented Programming

เทพพิทักษ์ การุญบุญญานันท์

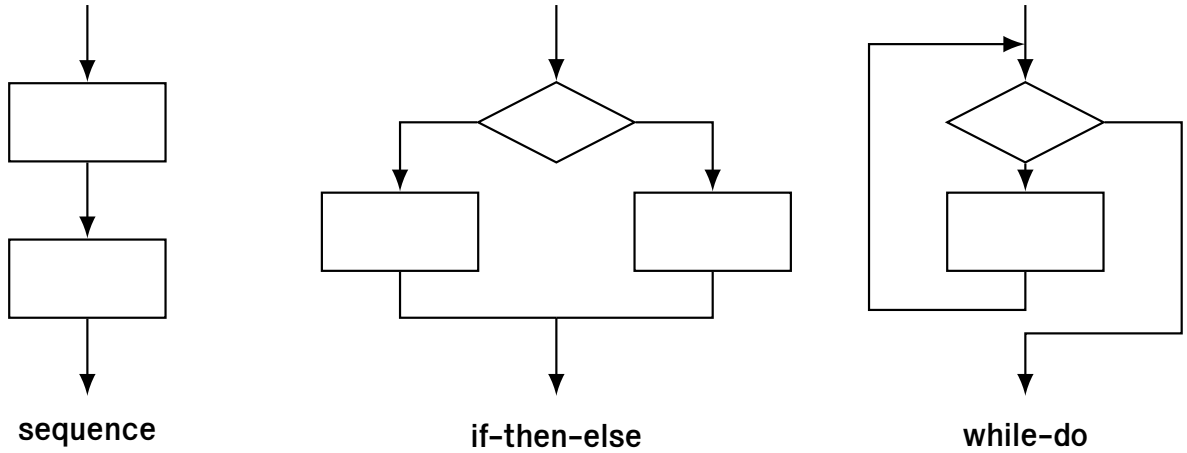
Programming Paradigms

- Non-structured Programming
  - มีแต่ branching และ goto อย่างง่าย
  - อนุญาตให้กระโดดไปที่บรรทัดใดก็ได้ในโค้ด
  - ข้อมูลมีแต่ชนิดพื้นฐานเท่านั้น

“Spaghetti Code!”

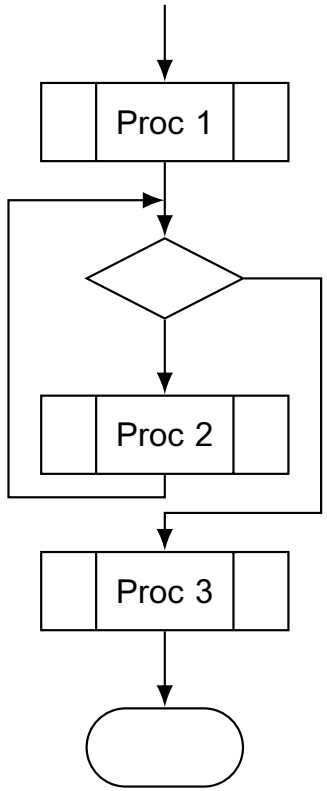


- Structured Programming
  - Edgster W. Dijkstra: “Go To Statement Considered Harmful” (CACM)
  - โค้ดที่ทำงานจากบนลงล่าง, เข้าทางเดียวออกทางเดียว
  - เลี่ยงการใช้ goto

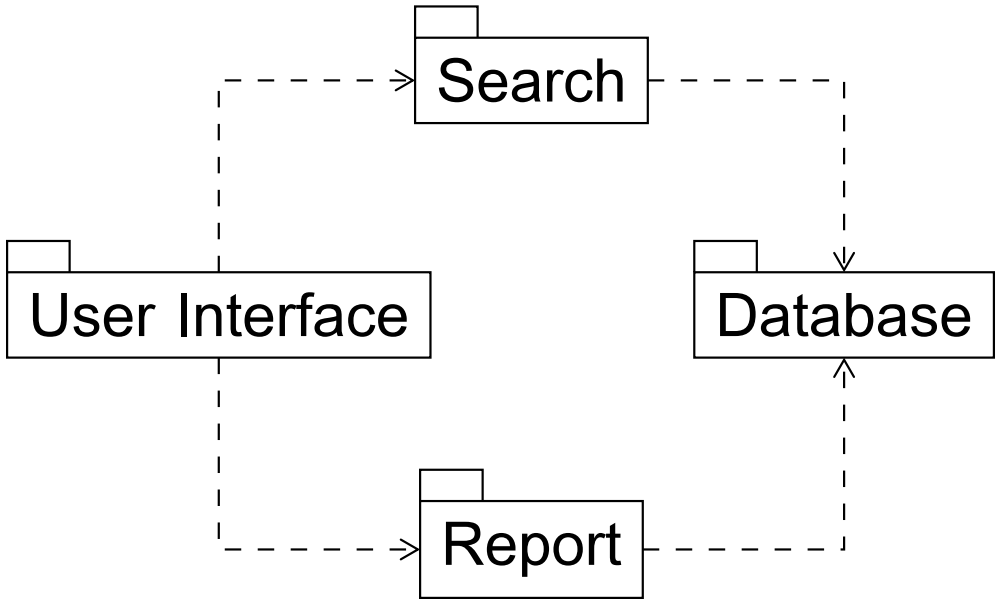


Programming Paradigms

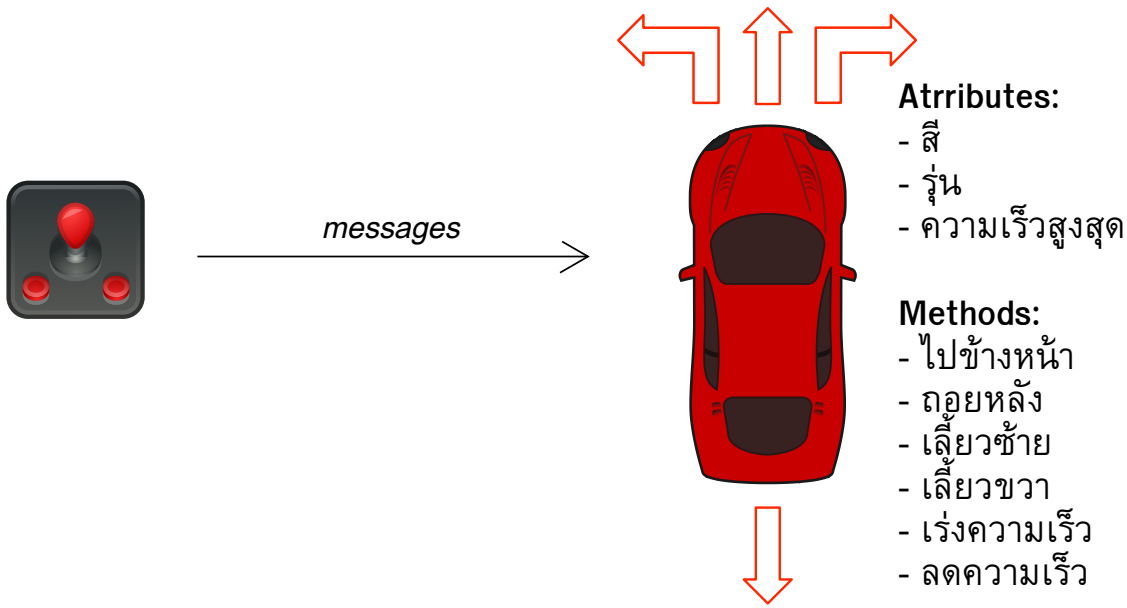
- Procedural Programming
  - พัฒนาจาก Structured Programming
  - แบ่งกระบวนการงานเป็นส่วน ๆ สำหรับเรียกใช้ เรียกว่า procedure
  - ส่วนหลักมองภาพรวมก่อน แล้วไปลงรายละเอียดใน procedure
  - top-down design หรือ stepwise refinement

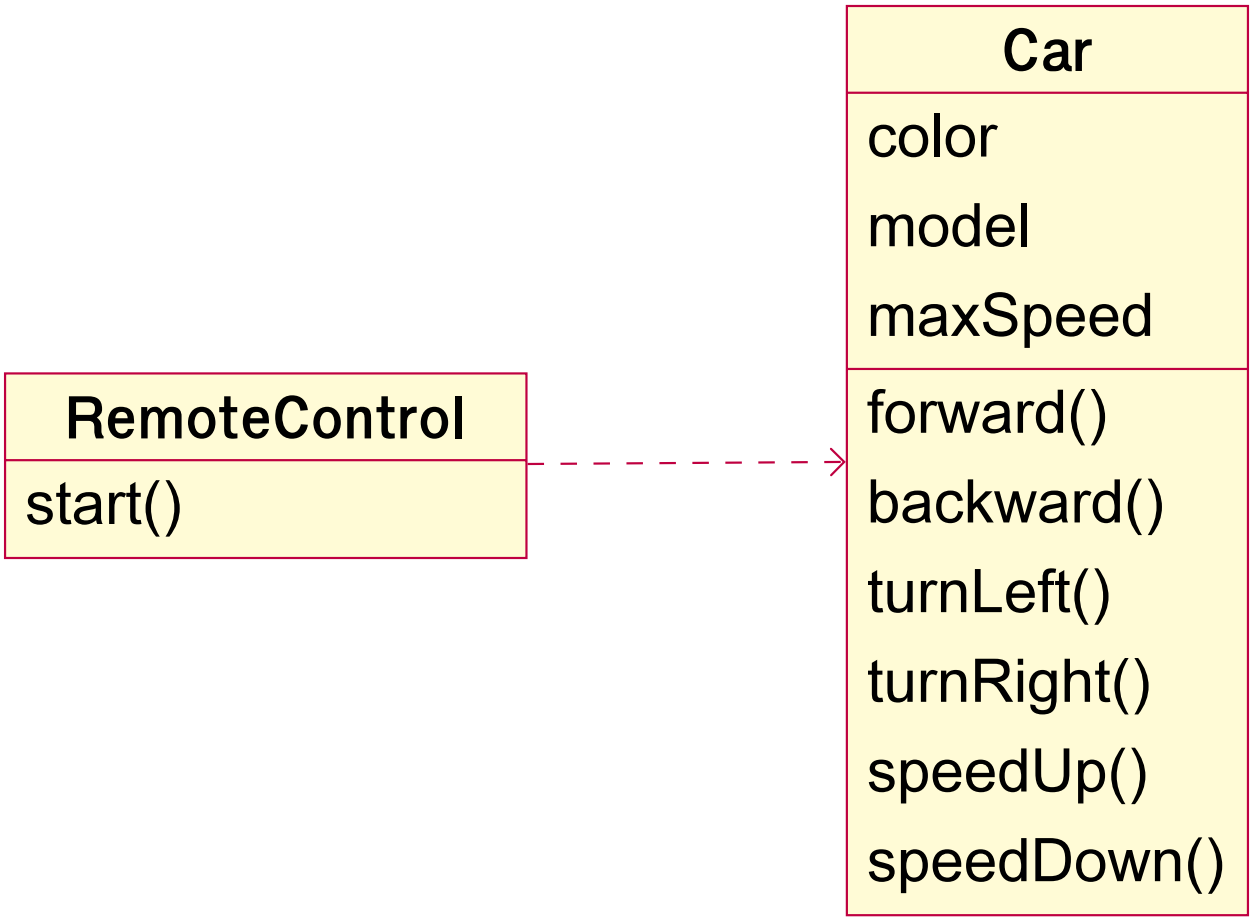


- Modular Programming
  - แบ่งโปรแกรมออกเป็นหน่วยย่อย ๆ ทำงานเป็นหมวด ๆ เรียกว่า module หรือ package
  - module = กลุ่มของ procedure ที่ร่วมกันทำงานหมวดหนึ่ง ๆ
  - module แยกอิสระจากกัน ทำงานร่วมกันผ่านอินเทอร์เฟซที่กำหนด



- Object-Oriented Programming
  - แบ่งโปรแกรมออกเป็นวัตถุ (object) ต่าง ๆ
  - วัตถุแต่ละชิ้นมีทั้งข้อมูลและ method (procedure) ให้เรียกใช้
    - → ทำกิจกรรมต่าง ๆ ได้ครบในตัวเอง
    - → คล้ายวัตถุในชีวิตจริง




Car
color
model
maxSpeed
forward()
backward()
turnLeft()
turnRight()
speedUp()
speedDown()
RemoteControl
start()

Procedural vs. Object-Oriented Programming

- Procedural Programming

กำหนดกระบวนการทำงานที่ต้องการ
ออกแบบโครงสร้างข้อมูลและอัลกอริทึมที่ดีที่สุดเพื่อทำงานให้สำเร็จ

- Object-Oriented Programming

กำหนดวัตถุที่ต้องการ
กำหนด operation ต่าง ๆ ของวัตถุแต่ละชิ้น
เชื่อมต่อวัตถุต่าง ๆ เข้าด้วยกัน

Procedural vs. Object-Oriented Programming

- โครงสร้างโปรแกรม
  - Procedural:
    - data structure และ procedure แยกจากกัน
    - ทุก procedure สามารถเข้าถึง data ได้ทุกชิ้นที่อยู่ใน scope
  - Object-Oriented:
    - data และ procedure ผูกมัดรวมกันเป็นก้อน ๆ
    - data ถูกกั้นไว้ไม่ให้เข้าถึงได้โดยตรงจากนอกก้อน
- เมื่อมีการเปลี่ยนแปลง data structure
  - Procedural:
    - ตามแก้ทุก procedure ที่เข้าใช้ data structure นี้
  - Object-Oriented:
    - แก้เฉพาะภายในก้อนที่เปลี่ยน โค้ดส่วนอื่นไม่กระทบ

ความซับซ้อนของซอฟต์แวร์

- ความซับซ้อนของตัวปัญหา (problem domain) เอง
  (requirements, usability, performance, cost, reliability)
- ความซับซ้อนของการจัดการงานพัฒนา
  (ขนาดของซอฟต์แวร์, การสื่อสารในทีม, การรักษา design)
- ความซับซ้อนจากความยืดหยุ่นของโลกเสมือนจริง
  (ซอฟต์แวร์สามารถสร้าง agent หรือ object ต่าง ๆ ได้ไม่จำกัดรูปแบบ)
- ความซับซ้อนของการดีบั๊ก
  (กรณีไม่คาดคิด, human error, การแก้บั๊กทำให้เกิดบั๊กใหม่)

ข้อจำกัดของมนุษย์

- จิตวิทยา: มนุษย์สามารถทำความเข้าใจข้อมูลพร้อม ๆ กันได้ไม่เกิน 7 (± 2) อย่าง
- สมองมนุษย์ใช้เวลาประมาณ 5 วินาทีในการรับข้อมูลชิ้นใหม่

$\rightarrow$ *เราต้องการเครื่องมือบริหารความซับซ้อน!*

Object-Oriented Programming Terminology

ออบเจกต์ (Object)

นามธรรมแทนบางสิ่งใน problem domain หรือใน implementation ซึ่งระบบสามารถเก็บข้อมูลของสิ่งนั้น ๆ และสามารถโต้ตอบกับสิ่งนั้น ๆ ได้

คลาส (Class)

คำบรรยายลักษณะของออบเจกต์ เป็นพิมพ์เขียวสำหรับสร้างออบเจกต์

Object = Class Instance

Object-Oriented Programming Concepts

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Abstraction

Abstraction

การละเลยรายละเอียดที่ไม่เกี่ยวข้องกับเป้าหมายปัจจุบัน
เพื่อมุ่งเน้นเฉพาะส่วนที่เกี่ยวข้องให้เต็มที่

ตัวอย่าง:

- ผู้ใช้โทรศัพท์มือถือรู้วิธีโทรออก/รับสาย โดยไม่จำเป็นต้องรู้กลไกภายใน
- ผู้ขับรถยนต์รู้วิธีเปลี่ยนเกียร์ โดยไม่จำเป็นต้องรู้กลไกภายใน
- รีโมทคอนโทรล

Abstraction

Abstraction เป็นแนวคิดทั่วไป:

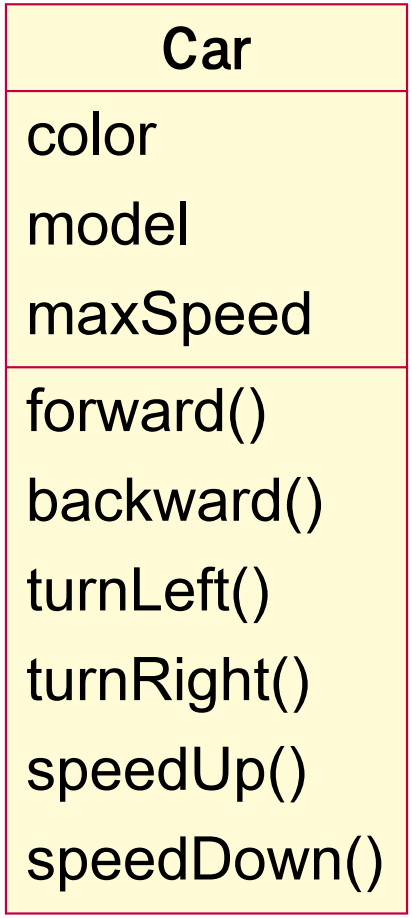
- Procedural: ละเลยรายละเอียดของ procedure ขณะเรียก
- Object-Oriented: เน้น data abstraction

Data Abstraction

การกำหนดชนิดของข้อมูลด้วยการกำหนด operation ต่าง ๆ ที่ทำกับข้อมูลชนิดนั้น ๆ โดยมีเงื่อนไขว่าค่าของข้อมูลดังกล่าวสามารถอ่านและเปลี่ยนแปลงได้ผ่าน operation เหล่านี้เท่านั้น

Abstraction

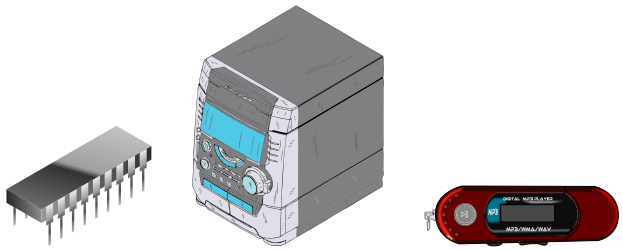
- Data Type = Attributes + Operations



Encapsulation

Encapsulation (Information hiding)

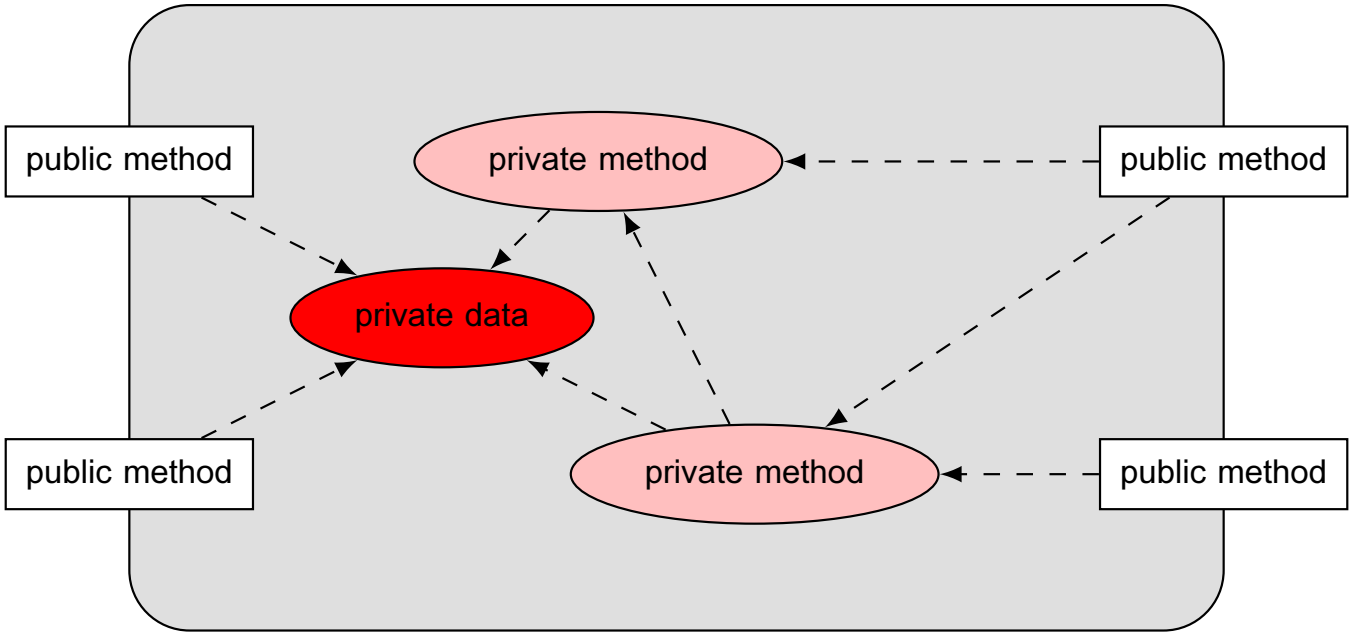
การออกแบบโปรแกรมโดยรวมสิ่งที่เกี่ยวข้องกันไว้ด้วยกัน
ซ่อนรายละเอียดของแต่ละองค์ประกอบไว้ภายใน
และเปิดให้เรียกใช้ผ่าน interface ที่กำหนดเท่านั้น



ตัวอย่าง:

- IC เก็บวงจรไว้ในกล่อง เปิดจุดเชื่อมต่อที่ขาต่าง ๆ ตามคู่มือเท่านั้น
- สเตริโอ เปิดช่องเสียบและปุ่มปรับต่าง ๆ เท่านั้น
- เครื่องเล่น MP3 เปิดช่อง USB สำหรับถ่ายโอนข้อมูล และปุ่มกดเท่านั้น

Encapsulation



ผล:

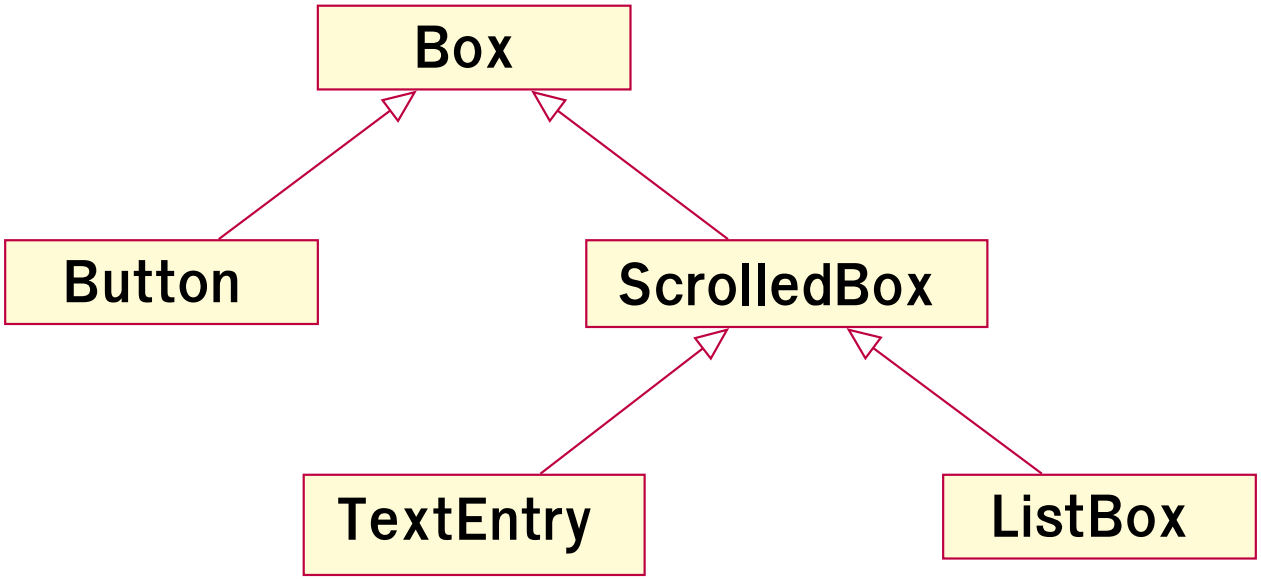
- จำกัดขอบเขตของงานหากมีการเปลี่ยนแปลง requirement
- ลดการจราจรระหว่างส่วนต่าง ๆ
- แยกผู้เรียก-ผู้ให้บริการออกจากกัน

Inheritance

กลไกการจัดแจงส่วนที่คล้ายกันระหว่างคลาสต่าง ๆ
โดยจัดลำดับชั้นของคลาสเป็นคลาสทั่วไปและคลาสเฉพาะ

- implement ส่วนที่ใช้ร่วมกันเพียงครั้งเดียว – ใน *base class*
- ต่อเติมเป็นกรณีเฉพาะเพิ่มได้ตามต้องการ – ใน *derived class*
- เป็นลำดับชั้นของความสัมพันธ์แบบ “is-a” คล้ายอนุกรมวิธาน (taxonomy)

ตัวอย่าง inheritance

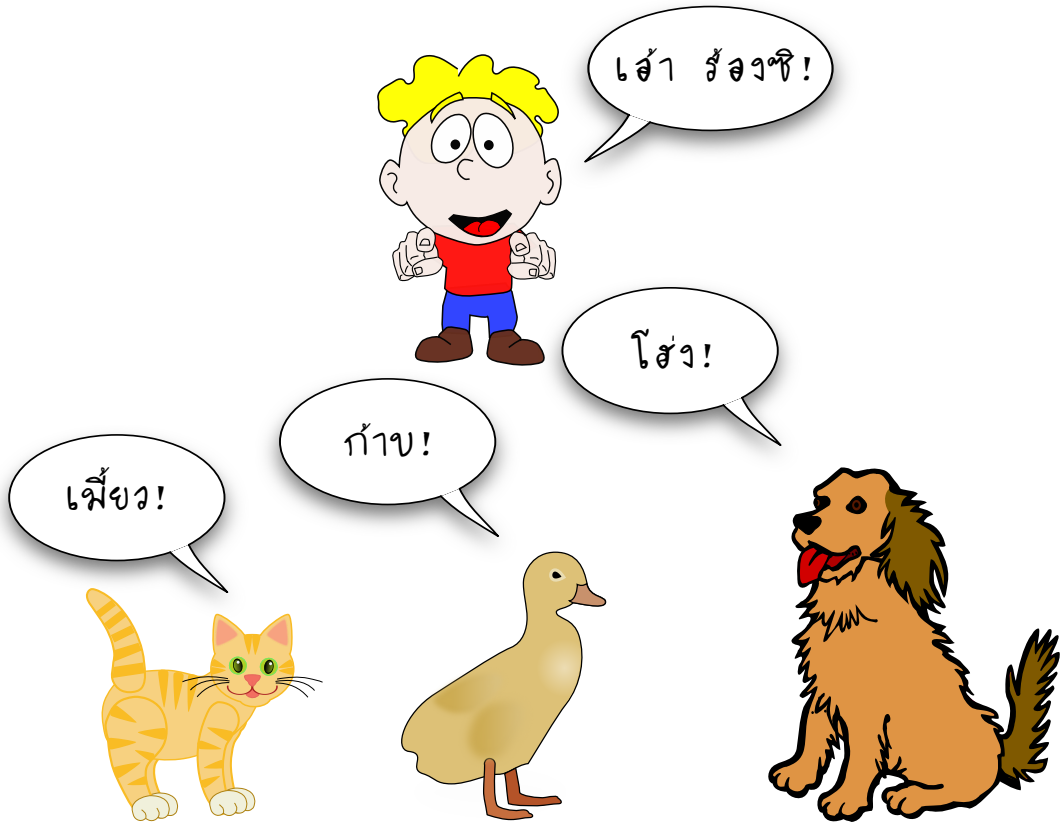


ผล:

- ลดการเขียนโค้ดซ้ำซ้อนในสิ่งที่ต้องใช้ร่วมกัน
- ต่อเติมเพิ่มคลาสใหม่ได้สะดวก
- เป็นเครื่องมือสำหรับ polymorphism (ดังจะกล่าวต่อไป)

Polymorphism

การกำหนด operation ที่มีพฤติกรรมได้หลากหลายตามกรณีเฉพาะต่าง ๆ

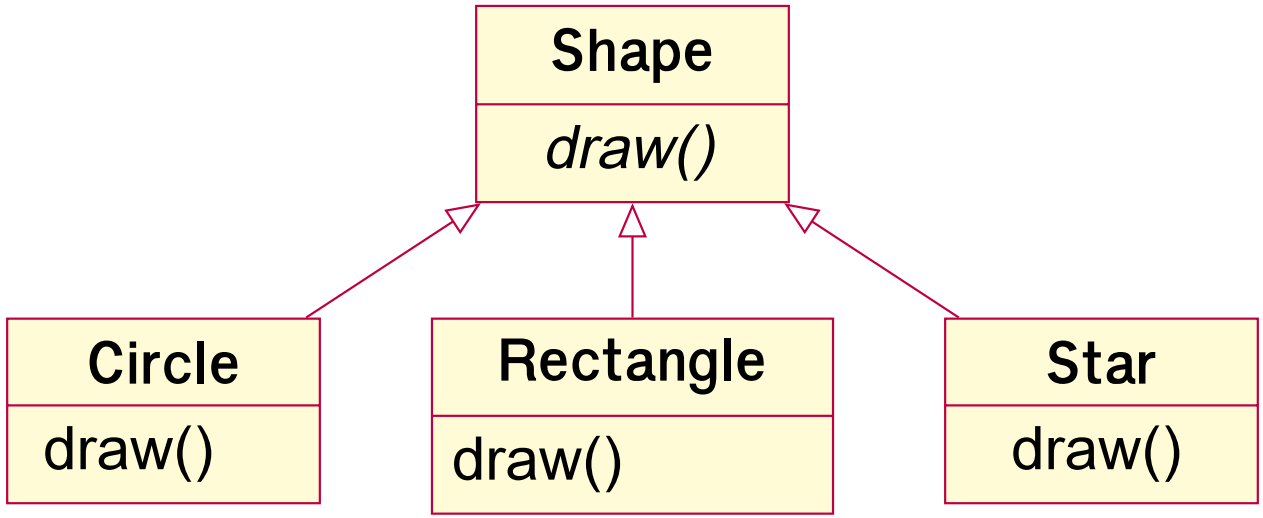


Polymorphism

- รากศัพท์กรีก polys (many, much) + morphe (form, shape)
- กำหนด operation รูปทั่วไปใน base class
- กำหนดพฤติกรรมรูปแบบเฉพาะใน derived class
- ผู้เรียก เรียก operation ทั่วไปนั้นผ่าน base class
  - → พฤติกรรมเป็นไปตาม derived class ของออบเจกต์นั้น ๆ

Polymorphism

ตัวอย่าง polymorphism



ผล:

- abstract การกระทำในระดับบน ลงรายละเอียดในระดับล่าง
- เพิ่มพฤติกรรมใหม่ ๆ ได้อย่างยืดหยุ่น โดยไม่ต้องแก้ไขโค้ดระดับบน
- เปิดแนวทางการออกแบบมอดูลในลักษณะ framework ซึ่งเป็นโค้ดระดับบน (แทนที่จะเป็นโค้ดระดับล่างให้เรียกใช้เหมือน library)