Software Design

อ. ปรัชญ์ ปิยะวงศ์วิศาล

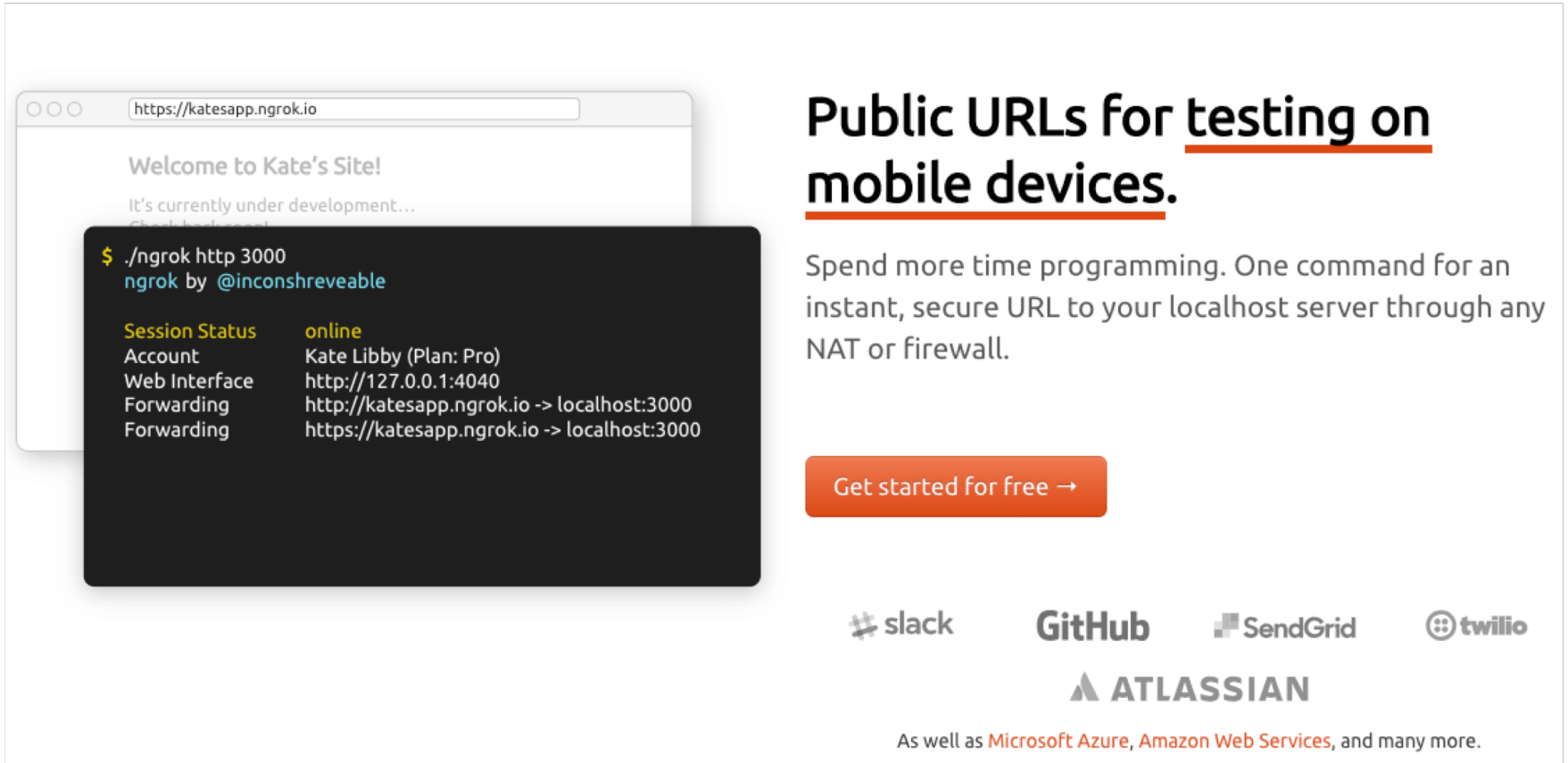
Pratch Piyawongwisal

Today’s Agenda

- Deploying Ionic App with ngrok
- OOP basics
- Software Design
  - Software Architecture
    - Server-Client
    - Model-View-Controller
    - Pipe-and-Filter
  - Design Patterns
    - Singleton, factory, adapter, observer
- Code Craftsmanship

Deploying Ionic App with ngrok

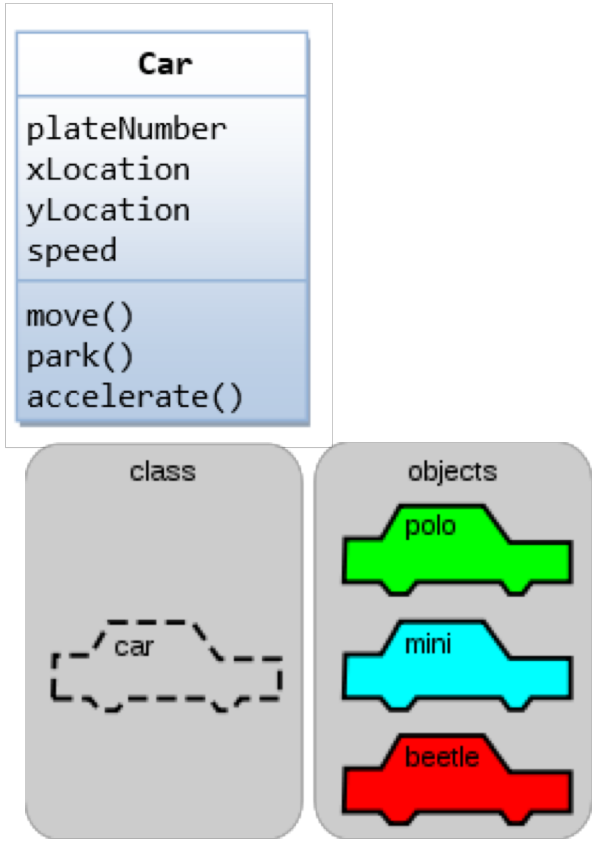
- โปรแกรมสำหรับ **deploy Ionic App,** ใช้งานง่ายกว่า **Heroku,** รันบนเครื่องตัวเอง
- ดาวน์โหลดได้ที่ https://ngrok.com/download



OOP Basics

Object-Oriented Programming (OOP)

- เป็นวิธีการเขียนโปรแกรมโดยใช้ **วัตถุ** (**object**) และ **คลาส** (**class**) ในการแก้ปัญหา
- เช่น จะเขียนโปรแกรมจองเที่ยวบิน เราจะเริ่มจากคิดว่าโปรแกรมควรมีวัตถุคลาสอะไรบ้าง
  - คลาส Flight เก็บข้อมูลเที่ยวบิน
  - คลาส Passenger วัตถุที่แทนตัวผู้โดยสาร
  - คลาส Destination ข้อมูลสนามบินปลายทาง
- คลาสประกอบด้วย 2 ส่วน คือ
  - Property คือ ส่วนเก็บข้อมูล
    - เช่น Passenger ต้องมี ID, Name, Address, RewardPoints, etc.
  - Method คือ ส่วนการกระทำ
    - เช่น Passenger ต้องสามารถ BookFlight(), ChangeProfile(),.. ได้



ตัวอย่างโค้ด OOP ในภาษา Java

1) การนิยามคลาส

Circle.java

```java
public class Circle {
    // properties
    double radius;
    String color;

    // methods
    double getRadius() {...}
    double printArea() {...}
}
```

2) การสร้างและใช้ object

***TestCircle.java* (MAIN)**

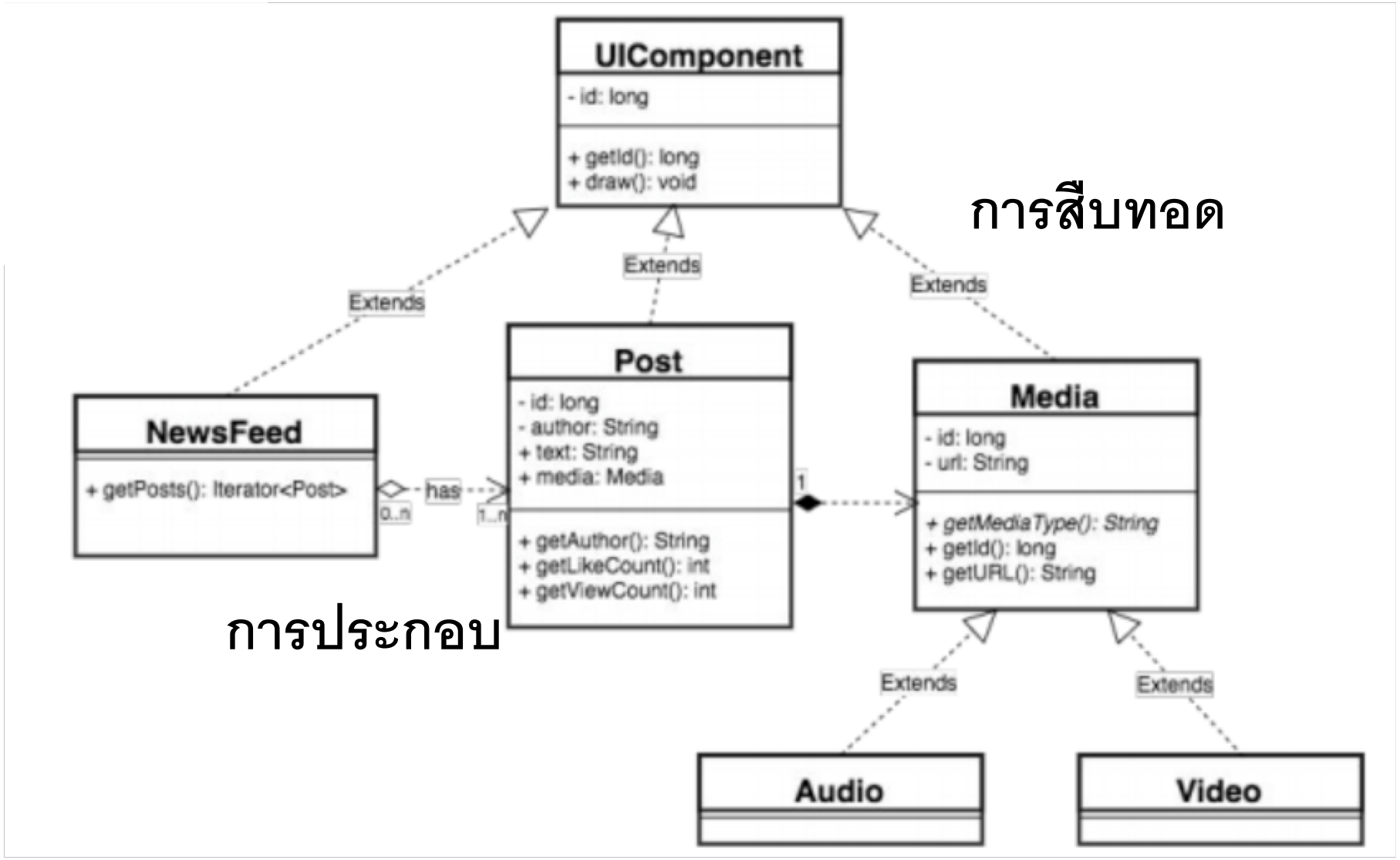
```java
public class TestCircle{
    public static void main(String[] args)
    {
        // objects instantiation
        Circle c1, c2;
        c1 = new Circle();
        c2 = new Circle();
        c1.radius = 2.0;
        c1.printArea();
    }
}
```

ตัวอย่างโค้ด OOP ใน Ionic (ภาษา TypeScript)

```typescript
export interface Medicine {
  name: String;
  description: String;
  amount: number;
  createdAt: number;
}
```

```typescript
medicine: Medicine = {
  name: 'Aspirin',
  description: 'ยาแก้ปวด',
  amount: 25,
  createdAt: new Date().getTime()
};
```

Inheritance และ Composition



Software Design

Software Design

- เป็นกระบวนการในการนิยาม architecture, components, interfaces ของระบบ
- ประกอบด้วย 2 ขั้นตอนหลักๆ ได้แก่
  - Software Architectural Design
    เป็นการออกแบบโครงสร้างของระบบในระดับ top-level โดยจะตอบคำถามที่ว่า เราควรแตกระบบออกเป็น component ย่อยๆ ได้อย่างไร?

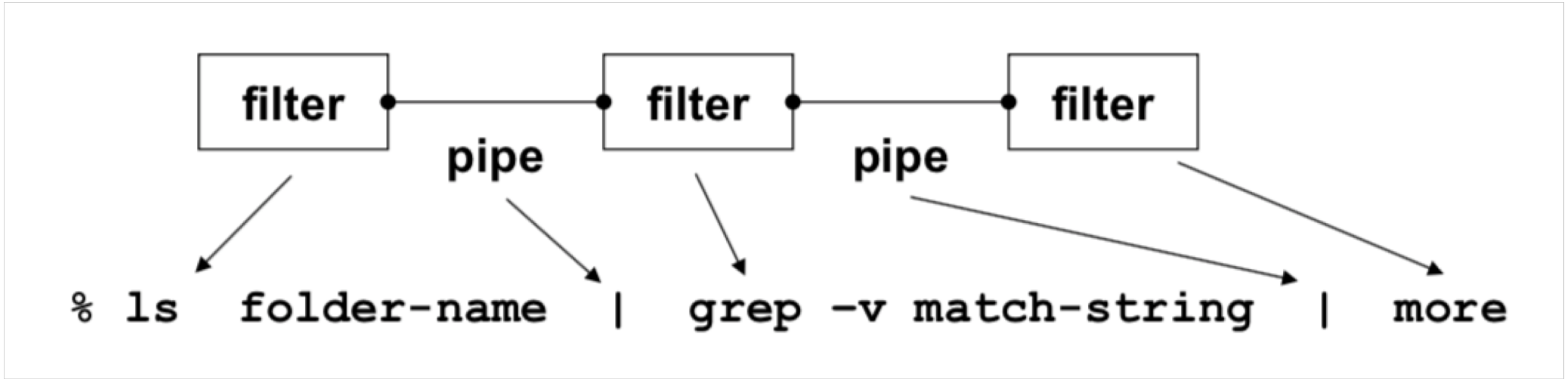
  - Software Detailed Design
    เป็นการออกแบบรายละเอียดภายในแต่ละ component ให้เพียงพอต่อการนำไปสร้างขึ้นมาได้จริง ==> design patterns

Software Architecture

- A software architecture is “a description of the **subsystems and components** of a software system and the **relationship** between them”
- ตัวอย่าง Software architecture ที่ควรรู้จัก
  - Pipe-and-Filter
  - Client-Server
  - Model-View-Controller

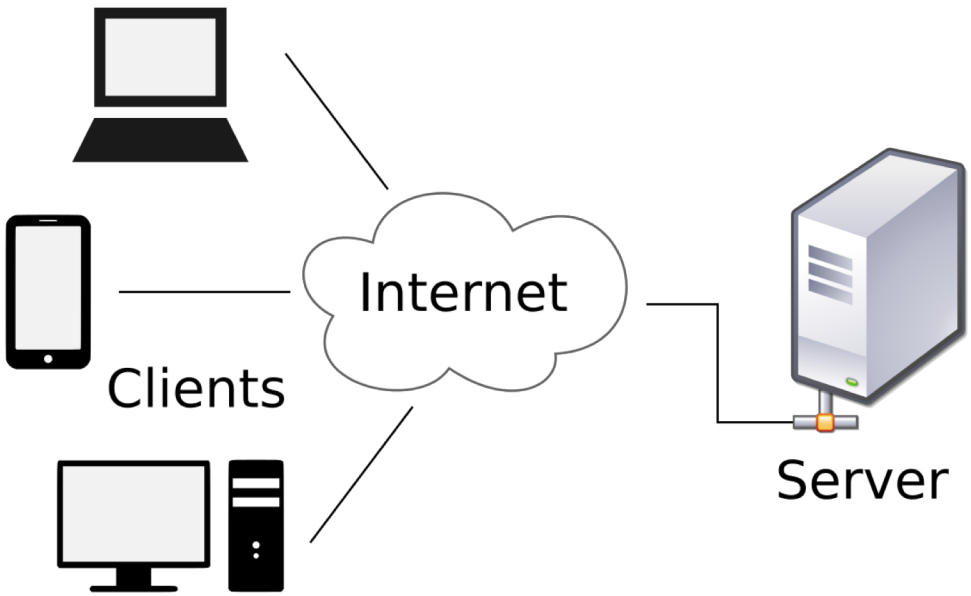
Pipe-and-Filter

- Components: **Filters** transform input into output
- Connectors: **Pipe** data streams
- Ex. UNIX shell commands



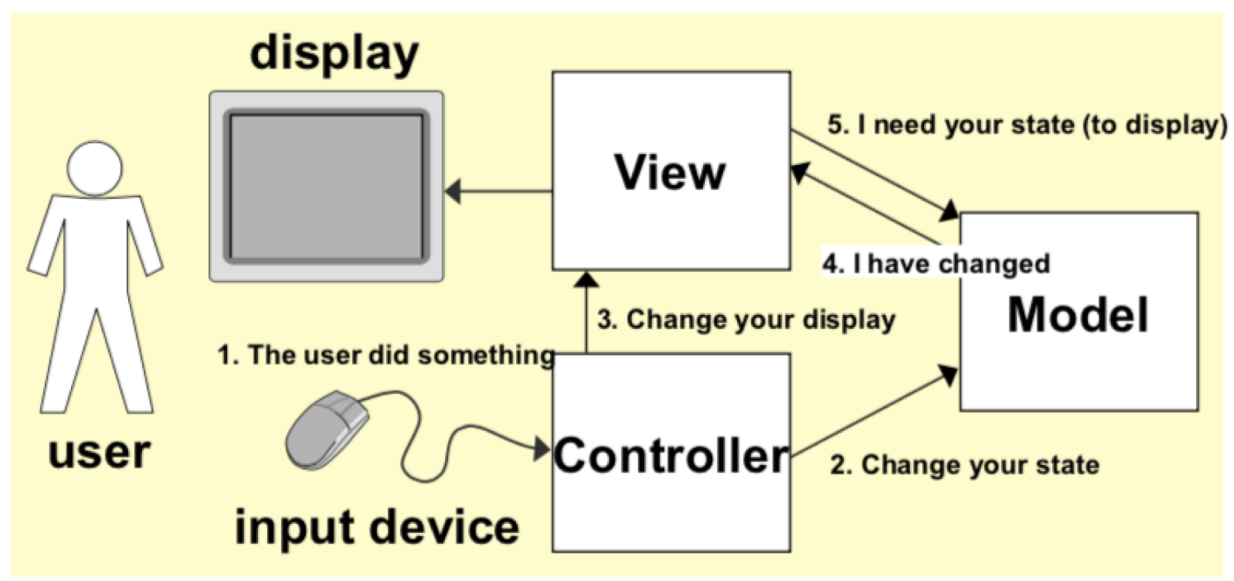
Client-Server

- Components: **Clients (ผู้ขอใช้บริการ)** and **Servers (ผู้ให้บริการ)**
- Connectors: **Protocols** และ **Messages** ที่ส่งไปมาระหว่างกัน



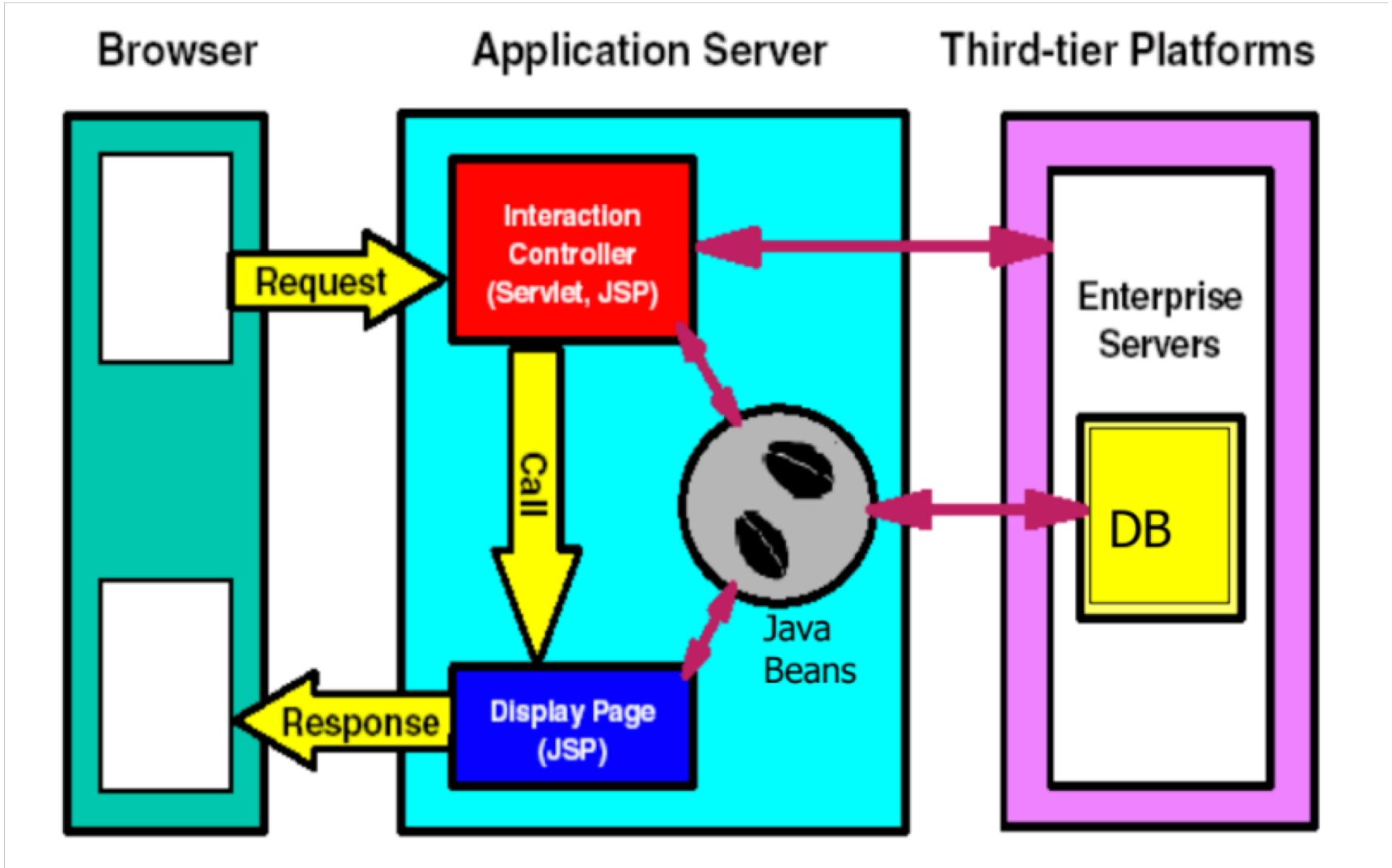
Model-View-Controller (MVC)

- ยึดหลัก Separation of Concerns แบ่งภาระหน้าที่ให้แต่ละส่วนทำงานไปโดยอิสระจากกันได้
- แบ่งระบบออกเป็น 3 ส่วน
  - Model ส่วนเก็บและจัดการข้อมูล เช่น โค้ดที่ใช้ติดต่อกับ Firebase, MariaDB
  - View ส่วนจัดวางรูปแบบการแสดงผล เช่น ไฟล์ .page.html ใน ionic
  - Controller ส่วนติดต่อกับผู้ใช้และ logic ควบคุม Model, View เช่น ไฟล์ .page.ts ใน Ionic
- นิยมใช้ในการเขียน Web/Mobile App
- ตัวอย่าง framework แบบ MVC
  - Ionic, Angular, Django, Ruby on Rails



Model-View-Controller (MVC)

อะไรคือ M, V, C ในภาพนี้ ?



Design Pattern

- Design Pattern is a “common design solution to a common problem in a given context”
- คือ รูปแบบการเขียนโปรแกรมที่พบเจอบ่อยๆ อาจเหมาะกับบางปัญหาตามแต่บริบท
- ตัวอย่าง Design Pattern ที่ควรรู้จัก
  - Singleton — บังคับให้แต่ละคลาสสามารถมีวัตถุได้แค่ตัวเดียวเป็นแบบ global/static
  - Factory
  - Adapter
  - Observer

} เป็น pattern ที่เจอบ่อย แต่ต้องเรียน OOP มาก่อนจึงจะเข้าใจ
ให้จำชื่อให้ได้พอ รายละเอียดการทำงานไม่ออกสอบ

Singleton

https://medium.com/20scoops-cnx/singleton-pattern-คืออะไร-b7b28182654f

```java
public class Singleton {
    private static Singleton instance;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

```java
public static void main(String[] args) {
        //init value
        UserData userData = UserData.getInstance();
        userData.setFistName("20Scoops");
        userData.setLastName("CNX");
```

ข้อดีคือ ป้องกันไม่ให้เกิดวัตถุจำนวนมากที่ทำงานซ้ำซ้อนกัน

ข้อเสียคือ เป็น global/static ถ้าเก็บข้อมูลใหญ่จะหน่วง

Code Craftmanship

Code Quality

- โค้ดที่มีคุณภาพ ที่แน่ๆ จะต้อง
  - correct ทำงานถูกต้อง
  - efficient มีประสิทธิภาพ (เร็ว, ใช้หน่วยความจำน้อย)
- นอกจากนี้ ยังจะต้อง
  - easy to read อ่านง่าย
  - easy to test ทดสอบง่าย
  - easy to deploy นำไปใช้ง่าย
  - easy to maintain ดูแลง่าย
- อีกด้วย

Why care about code quality?

- ปัญหาอันเกิดจากโค้ดที่มีคุณภาพต่ำ
  - productivity ต่ำ ทำให้บริษัทเติบโตช้า
  - ส่งผลกระทบต่อ user experience (เกิด bug, error)
  - ทำให้โปรแกรมเมอร์เก่งๆ ไม่อยากมาทำงานด้วย
    - คุณภาพโค้ดก็จะต่ำลงไปอีก

How to?

- เขียนโค้ดอย่างไรให้มีคุณภาพ?
- Craftsmanship – ศิลปะในการเขียนโค้ดที่มีคุณภาพ
  - Attention to details ใส่ใจกับรายละเอียด ไม่สุกเอาเผากิน
  - Refactoring ปรับโครงสร้างโค้ดให้ดีขึ้นตลอดเวลา
  - Consistency ความสม่ำเสมอของ coding style/convention
- Tools
  - ใช้ tool มา automate กระบวนการต่างๆ
- Communication
  - มีการทำ peer review

Basic rules

- ใช้ coding style/convention ตามโค้ดที่มีอยู่เดิมใน codebase
  - ตำแหน่งปีกกา
  - จำนวน space ระหว่างเครื่องหมายอย่าง , ; =
  - การตั้งชื่อตัวแปรและฟังก์ชัน (num_items vs numItems)
- เขียนโค้ดให้รวบรัด แต่ก็ยังคงความอ่านง่าย
- ตั้งชื่อตัวแปรให้บ่งบอก ไม่ใช้ชื่อแปลกๆ เช่น tt, aba, image2, num_st
- พยายามไม่ให้เกิดโค้ดที่ซ้ำซ้อน duplicate
- ใส่ comment อธิบายโค้ด
  - เขียนให้กระชับ ได้ใจความ ไม่เวิ่นเว้อ
  - หากนำโค้ดมาจาก stackoverflow ใส่ url ที่มาด้วย
- พยายามไม่ comment code ทิ้ง เพราะรก

Good vs bad coding styles

- Examples:

  U Toronto CSC301 Code Craftsmanship slides

  - https://drive.google.com/file/d/1sPOba8be7eSynG_aBnoemEDfHxgrMTAT/view

Code Quality Tools

- โปรแกรม lint
  - ใช้หาโค้ดส่วนที่ผิดปกติอย่างเห็นได้ชัด
    - ตัวแปรที่ไม่ได้ใช้
    - โค้ดที่ไม่เคยรัน
    - เงื่อนไขที่เป็นจริงเสมอ
    - คำสั่งที่ไม่ทำให้เกิดอะไร (no effect)
- โปรแกรม style checker
  - IDE ดีๆ อย่าง VS Code, Eclipse มักมีให้
    - auto-indent
    - auto-format
  - สามารถใช้ travis CI ในการ automate

Peer Review

- Code review
  - คือการให้เพื่อนร่วมงานช่วยตรวจทานโค้ดก่อนการ merge
  - นิยมทำกันทั่วไปในบริษัทซอฟต์แวร์
  - โปรแกรมเมอร์อาวุโสช่วยดูให้โปรแกรมเมอร์มือใหม่
  - โค้ดผ่านหลายตา ย่อมเจอจุดผิดได้มากขึ้น
  - ให้ feedback ที่เป็นประโยชน์ ใช้คำวิจารณ์ที่ตรงแต่ถนอมน้ำใจ
- เป็นตามค่านิยมแบบ Agile คือส่งเสริมให้เกิดการพูดคุยแลกเปลี่ยน

อ่านเพิ่มเติม

- Style Guide สำหรับโปรแกรมเมอร์ที่ทำงานที่ Google
  - https://google.github.io/styleguide/cppguide.html
  - https://google.github.io/styleguide/javaguide.html
- หนังสือ Clean Code
  - https://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882