UML Extension for Web Service Design

(UML API Diagram)

OMG Standards Development Organization

ABOUT US | GROUPS | CERTIFICATIONS | RESOURCES | SPECIFICATIONS | COMMUNITIES | MEMBERSHIP

UML PROFILE CATEGORY - SPECIFICATIONS ASSOCIATED

UML PROFILE • CATEGORIES • SPECIFICATIONS

CATEGORY Uml Profile

This page provides a summary of OMG specifications that have either been formally published or are in the finalization process.

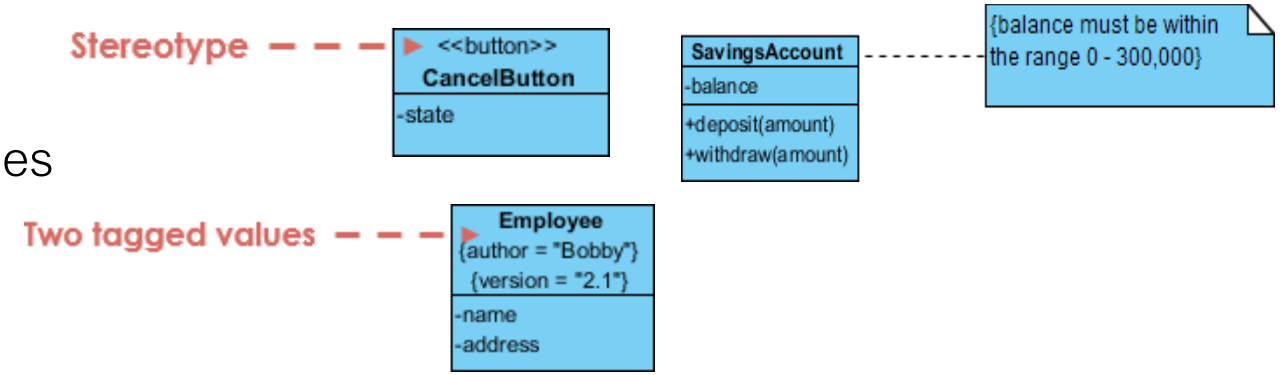
The "acronym" link navigates to the latest version of the specification, this link changes whenever a new version of the specification is published.
The "Version" link navigates to the specific version.

Search inside this table

NAME	ACRONYM	VERSION	STATUS	PUBLICATION DATE
UML Profile for BPMN Processes	BPMNProfile™	1.0	formal	July 2014
UML Profile for CORBA and CORBA Components	CCCMP™	1.0	formal	April 2008
UML Profile for CORBA Components	CCMP™	1.0	formal	July 2005

Profile Diagram (UML Profile)

- Profile diagram is basically an extensibility mechanism that allows you to extend and customize UML by adding new building blocks, creating new properties and specifying new semantics in order to make the language suitable to your specific problem domain
- Profile diagram has three types of extensibility mechanisms:
  - Stereotypes
  - Tagged Values
  - Constraints



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-profile-diagram/

UML Profile for REST API Design

- Several UML profile practices are available
  - UML profile for RestAPI by SparxSystem (เน้นวิธีนี้และปรับเพิ่มได้)
  - UML profile for RestAPI by IBM
  - UML profile for RestAPI by Papyrus

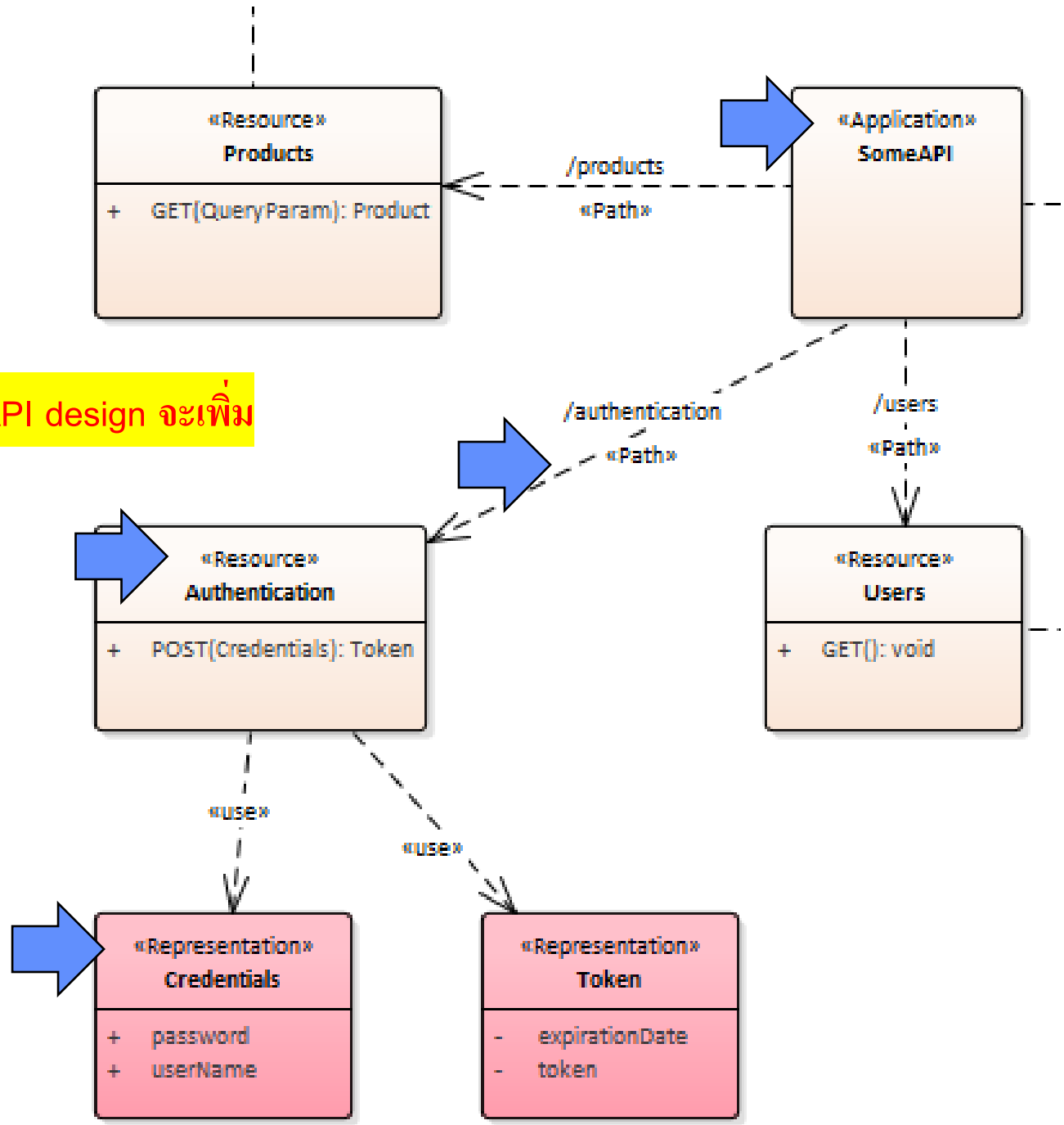
UML profile for REST API design จะเพิ่ม

<<Application>>

<<Path>>

<<Resource>>

<<Representation>>



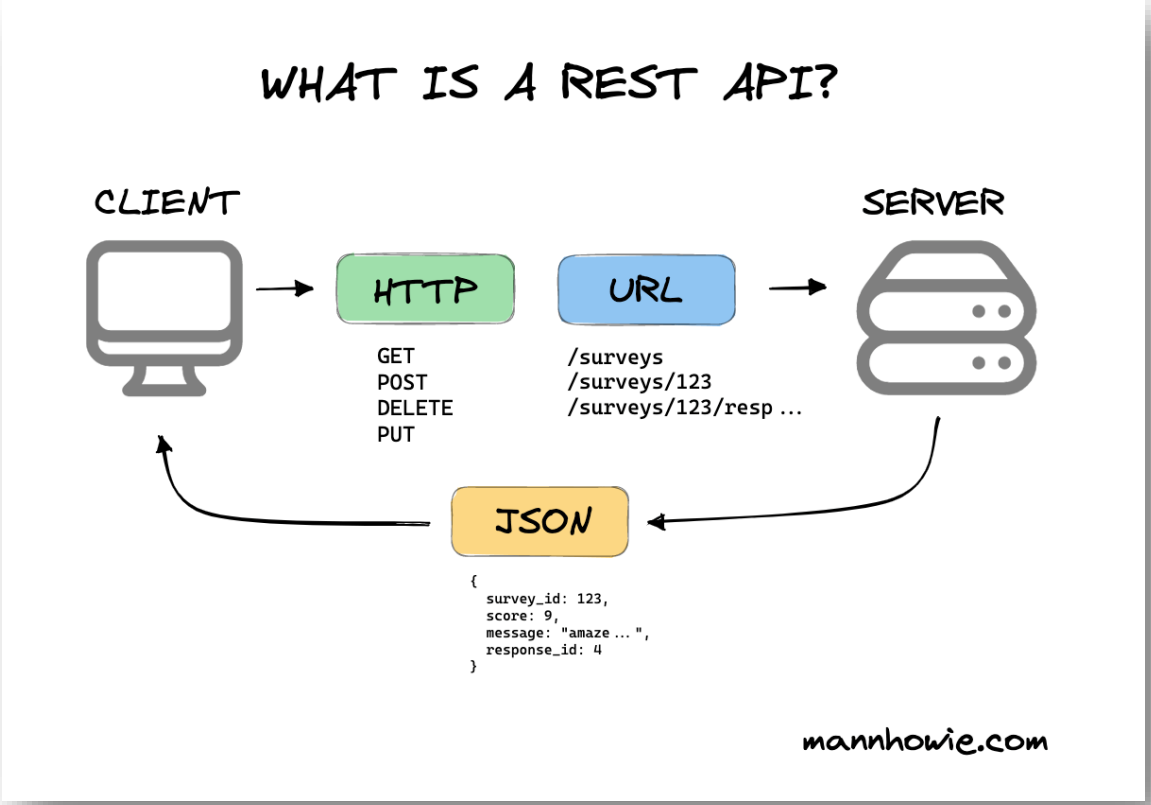
REST API using HTTP Requests

HTTP Methods

- ★ GET
- ★ POST
- ★ PUT
- HEAD
- ★ DELETE
- ★ PATCH
- OPTIONS
- CONNECT
- TRACE

PUT Totally update

PATCH Partially update



GET Method

- The GET Method
  - GET is used to request data from a specified resource.
  - Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.php?name1=value1&name2=value2
```

Query string

จงใช้อย่างระวัง

https://www.w3schools.com/

ข้อควรระวังเกี่ยวกับ GET

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)

https://www.w3schools.com/

POST Method

- The POST Method
  - POST is used to send data to a server to create/update a resource.
  - The data sent to the server with POST is stored in the request body of the HTTP request:

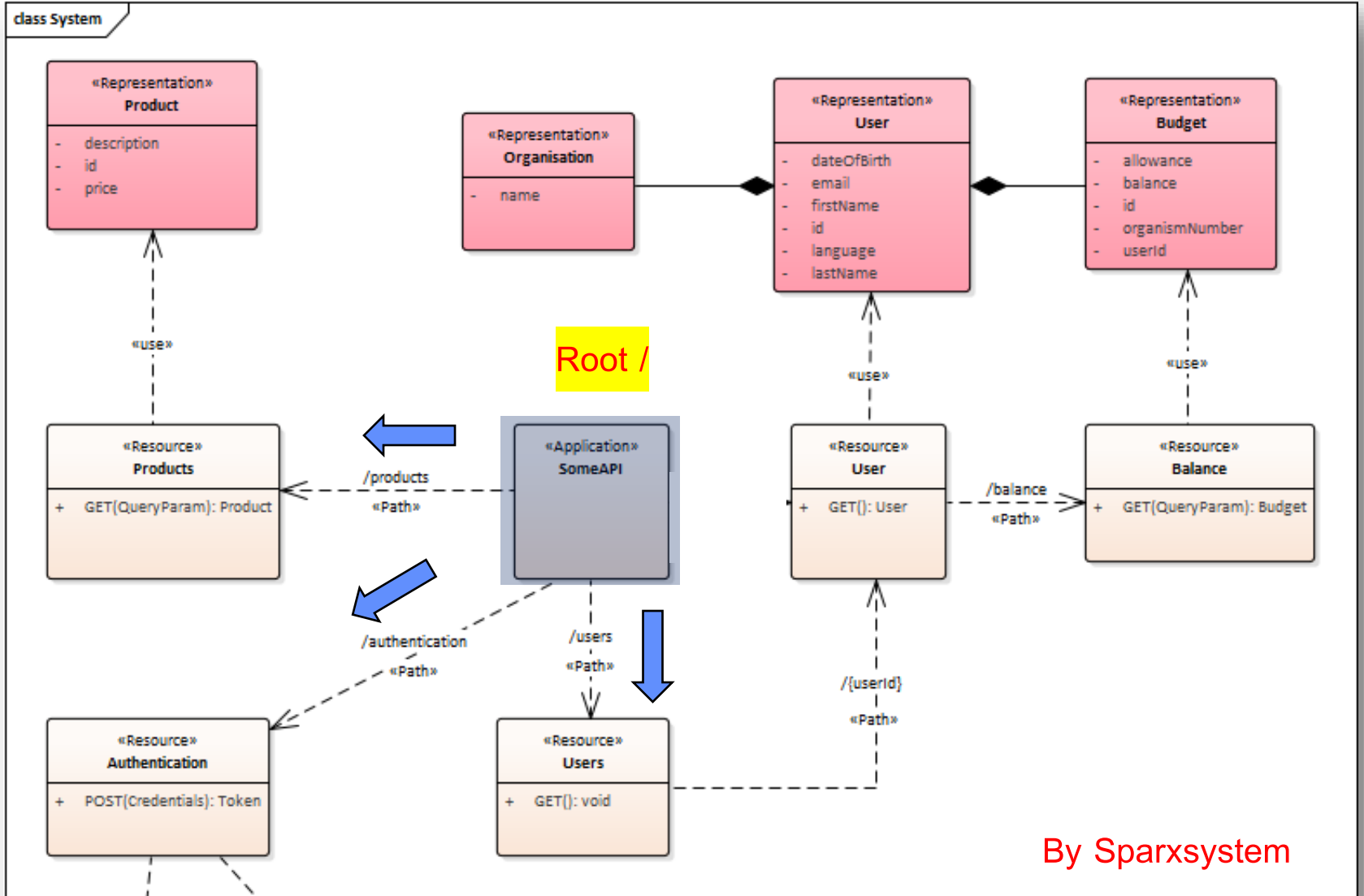
```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com

name1=value1&name2=value2
```

Query string

https://www.w3schools.com/

ข้อควรระวังเกี่ยวกับ GET

- POST requests are never cached
  - ข้อเสียของ POST หลัก ๆ คือ ถ้าเรากด back button/reload จะมีการ re-submitted request ต้องระวังเตือนให้ผู้ใช้ทราบเสมอ
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

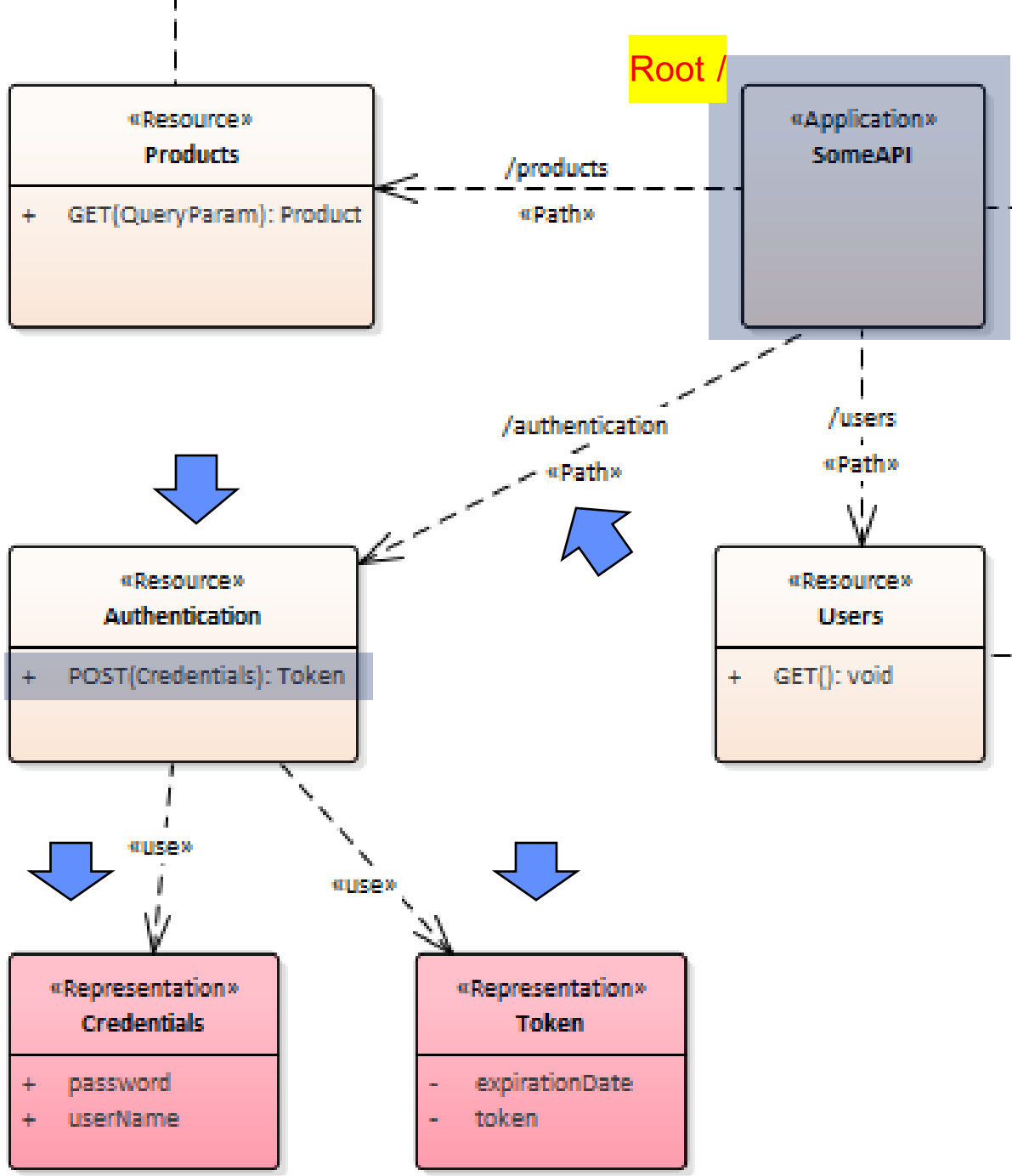


To model the API you create a new class diagram and start by the root of your API as a class of stereotype "Application". Next you add your Resources and Paths. For each Resource you add the necessary GET, POST, PUT etc. operations. If QueryParams are passed along in the string for the GET operations you add them as Parameters for the operation, but as type QueryParam (you need to create it the first time). For the messagebody of POST operations you add a parameter of the class type it represents. https://sparxsystems.us/go/restapi/

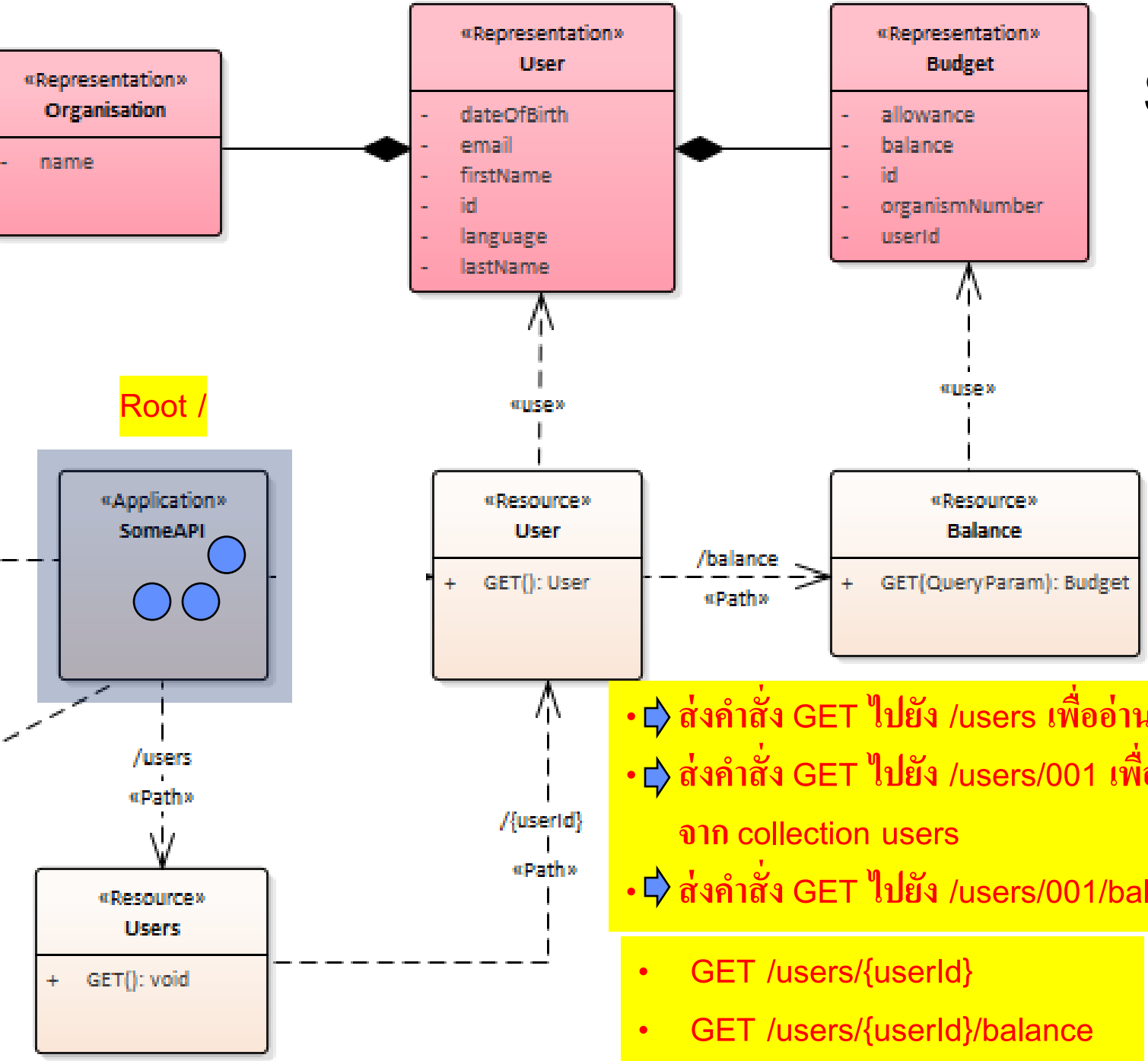
Next you add the "representation" classes, used by the Resources.

Sample of REST API

- <<Resource>> ระบุทรัพยากรและบริการที่มี
- <<Path>> ระบุเส้นทางเข้าถึงทรัพยากร
- <<Representation>>ระบุ entity/model ที่เกี่ยวข้อง
- ส่งคำสั่ง POST ตามเส้นทาง /authentication เพื่อ…
- ส่งคำสั่ง GET ไปยัง /products เพื่อ…



Sample of REST API



- ส่งคำสั่ง GET ไปยัง /users เพื่ออ่าน collection of users
- ส่งคำสั่ง GET ไปยัง /users/001 เพื่ออ่าน user id 001 จาก collection users
- ส่งคำสั่ง GET ไปยัง /users/001/balance เพื่อ…

- GET /users/{userId}
- GET /users/{userId}/balance

ตัวอย่างการ document API ที่จัดทำเพื่ออ้างอิง (สร้างโดย Swagger ก็ได้)

Salesforce B2C

Search: Go Scope: All topics

RoleSea
Settings
SitePref
Sites res
SiteSea
SlotCon
SlotCon
SlotCon
Slots res
SlotSea
SortingF
SourceC
SourceC
Stores r
StoreSe
System
System
Users re
UserSea
VariantS
Shop API d
Data API d
Script Debugge
Commerce Clo
Glossary
B2C Commerc
Storefront Refe
SiteGenesis Se

Salesforce B2C Commerce 23.3 > Open Commerce API 23.1 > Data API resources 23.1

Users resource (Data API 23.1)

Summary

Http Method	Resource	Description
GET	/users	Action to get all users with no filtering.
GET	/users/this	Action to get the user password expiration information.
PATCH	/users/this/password	Action to change a local user password. This action cannot change the password for a user managed by Account Manager.
GET	/users/{login}	Action to get a user.
PUT	/users/{login}	Action to create or overwrite a user. If a user with the given login already exists, the existing user will be overwritten. If no such login exists, a new user is created.
PATCH	/users/{login}	Action to update a user. Note: The `locked` flag and the user `password` cannot be updated with this resource.
DELETE	/users/{login}	Action to delete a single user.

Design นั้น สำคัญไฉน

- Design = Arch. Design (High level) + Detailed Design (Low level)
- Agile Dev. เช่น SCRUM เรา document design เท่าที่จำเป็น
- ถ้าเป็น Team Dev ขนาดใหญ่ เรายิ่งควรจะมี Design Doc

Detailed Design

(Object Detailed Design)

Modified from Bernd Bruegge and Allen H. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd Edition, Pearson, 2013.

Object Detailed Design

- Object design is the process of adding details to the requirements analysis and making implementation decisions
- The object designer must choose among different ways to implement the analysis model with the goal to minimize execution time, memory and other measures of cost.
- Object design serves as the basis of implementation

Specifying Interfaces

- Analysis and architectural design activities ก่อนหน้า
  - Identifying attributes and operations without specifying their types or their parameters.
- Object design activities ช่วงหลัง
  - identifying missing attributes and operations
  - specifying visibility and signatures
    - decide which operations are available to other objects and subsystems, and which are used only within a subsystem
    - specify return type of each operation as well as the number and type of its parameter
  - specifying contracts
    - describe constraints or conditions that must be met before the operation is invoked and a specification of the result after the operation returns

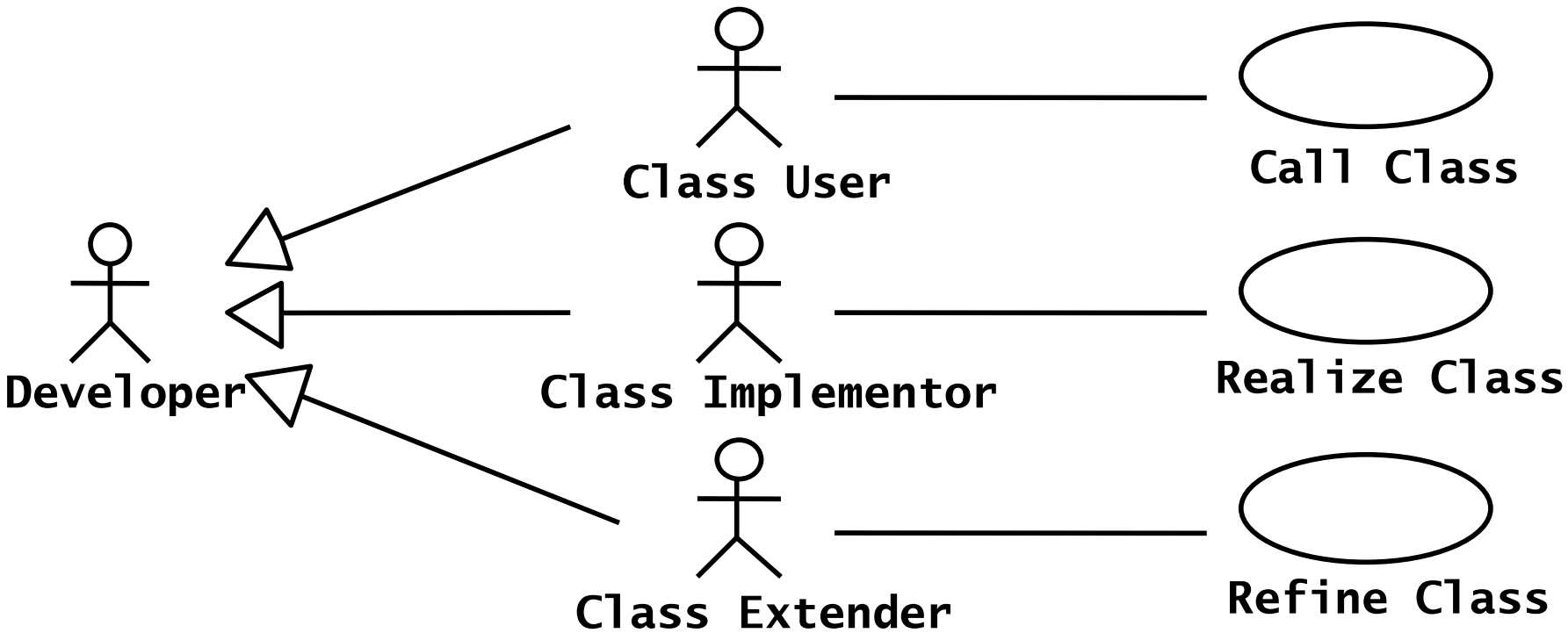
Interface Specification Concepts

- Class Implementor, Class Extender, and Class User
- Types, Signatures, and Visibility
- Contracts: Invariants, Preconditions, and Postconditons

Class Implementor, Class Extender, and Class user

- Class Implementor
  - realizes the class under consideration
  - designs the internal data structures
  - implements the code for each publication operation
- Class User
  - invokes the operations provided by the class under consideration during realization of another class, called **client class**
  - discloses the boundary of the class in terms of the services it provides and the assumptions it makes about the client class
- Class Extender
  - develops specializations of the class under consideration
  - focuses on specialized versions of the same services

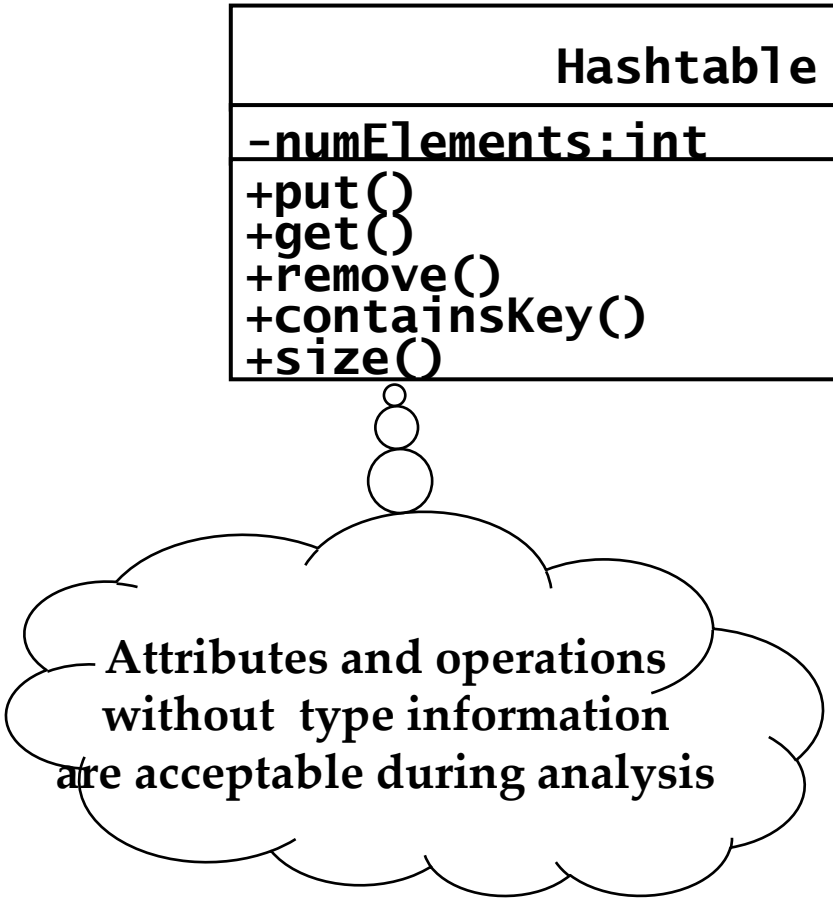
The Class Implementor, the Class Extender, and the Class User role

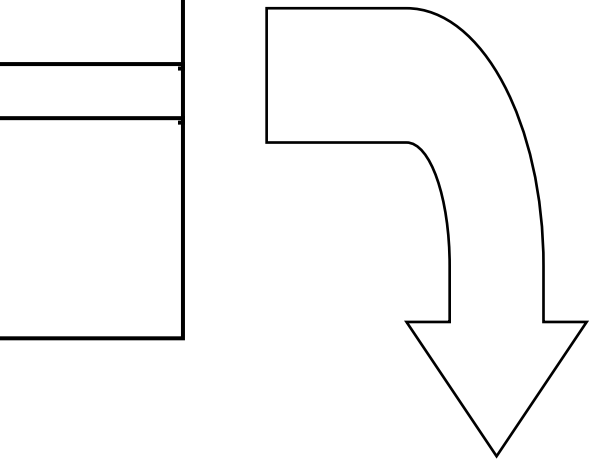


Types, Signatures, and Visibility

- **Type** of an attribute
  - range of value the attribute can take
  - operations that can be applied to the attribute
- Signature
  - type of Operation parameters and type of return values
  - define range of values the parameter or the return value can take

Add Type Signature Information





Hashtable

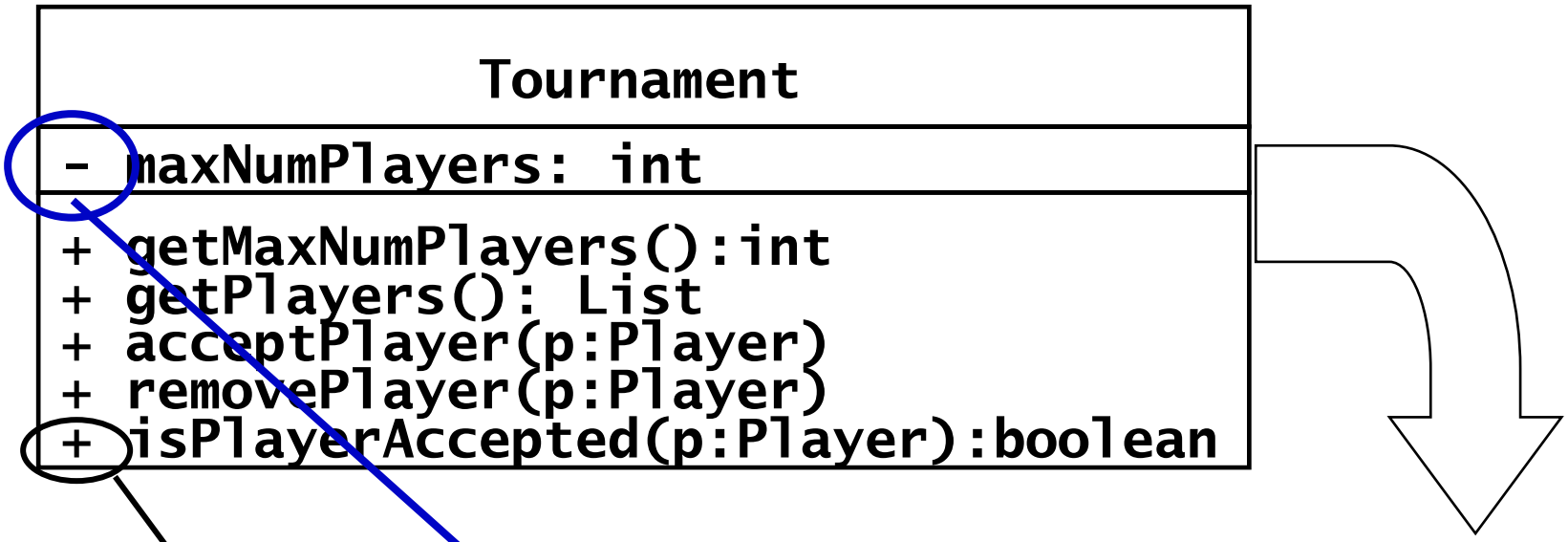
-numElements:int

+put(key:Object,entry:Object)
+get(key:Object):Object
+remove(key:Object)
+containsKey(key:Object):boolean
+size():int

Types, Signatures, and Visibility (cont.)

- **Visibility** is a mechanism for specifying whether the attribute or operation can be used by other classes or not.
  - A private attribute (similar to a private operation)
    - can be accessed only by the class in which it is defined
    - for class implementor only
  - A protected attribute or operation
    - can be accessed by the class in which it is defined and by an descendent of that class
    - for class extender
  - A public attribute or operation
    - can be accessed by any class
    - for class user

Implementation of UML Visibility in Java

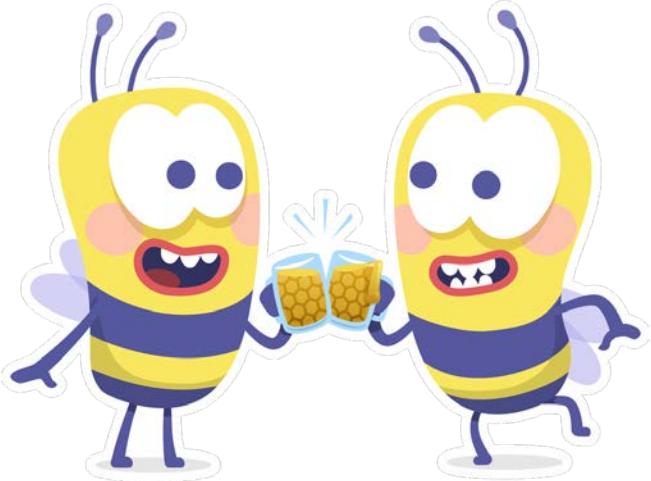



```
public class Tournament {
    private int maxNumPlayers;

    public Tournament(League l, int maxNumPlayers)
    public int getMaxNumPlayers() {…};
    public List getPlayers() {…};
    public void acceptPlayer(Player p) {…};
    public void removePlayer(Player p) {…};
    public boolean isPlayerAccepted(Player p) {…};
```

Contracts

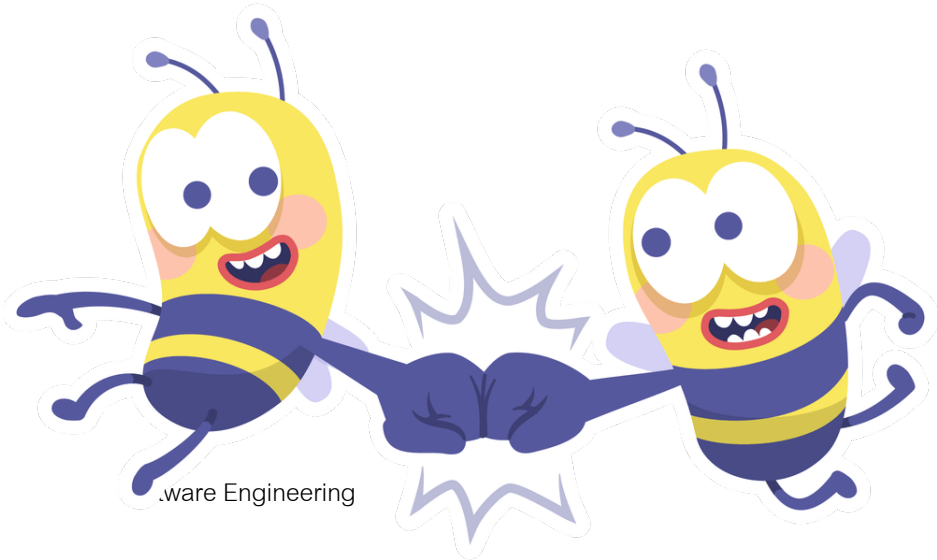
- Contracts are constraints on a class that enable class users, implementors, and extenders to share the same assumption about the class
- Contracts include 3 types of constraints
  - invariant is a predicate that is always true for all instances of a class
  - precondition
    - is a predicate that must be true before an operation is invoked
    - is associated with a specific operation
    - is used to specify constraints that a class user must meet before calling the operation
  - postcondition
    - is a predicate that must be true after an operation is invoked
    - is associated with a specific operation
    - is used to specify constraints that the class implementor and the class extender must ensure after the invocation of the operation

Design by Contract



Frontend Dev. กับ Backend Dev.

ต้องคุยกันอย่างไร

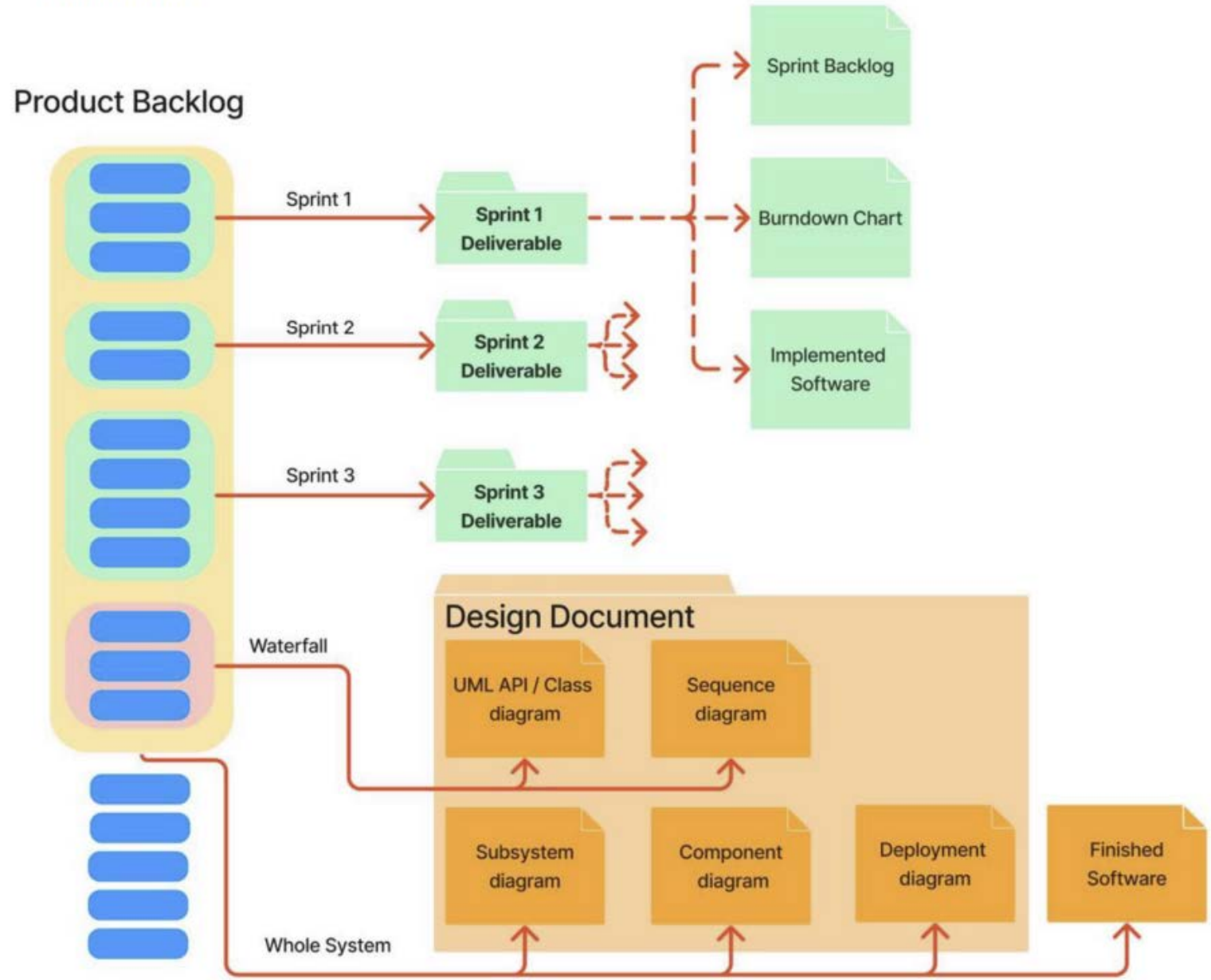


Term project

SW Design Documents for Term Project

- Selected SW feature for Waterfall process called “feature W”
- UML design diagrams to be submitted for “feature W” are:
  - Subsystem/Component diagram of the whole system
  - Deployment diagram of the whole system
  - Class diagram of “W” only
    - If REST APIs are needed, then draw UML profile for REST API
  - Sequence diagram of “W” only
- Source code of “feature W” will be traceable back to design diagrams

2. Architecture

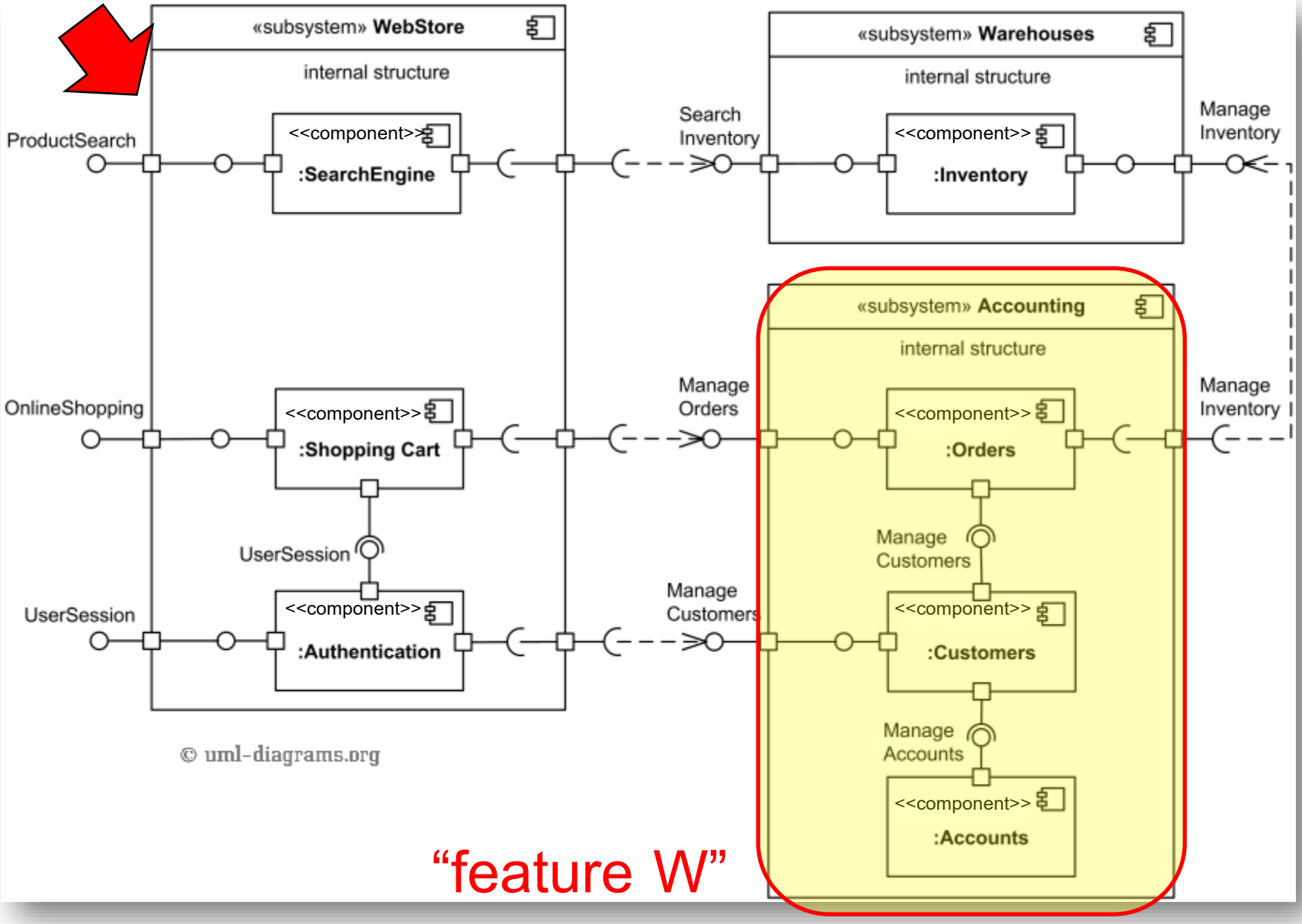


- For the whole system (S1+S2+S3+Waterfall)
    - Subsystem diagram
    - Component diagram
    - Deployment diagram
- For the selected features (Waterfall)
    - Sequence diagram (one use case)
    - Each group must submit **either**
        - Class diagram including Attributes, Operations, Visibility (if using OO design) **or**
        - UML API diagram (if using web development pattern)

3. Access Control Table **or** API CRUD

The rest features are from SCRUM

ตัวอย่าง subsystem/component diagram



"feature W"

Access Control

- Different actors have access to different functionality and data
- During analysis
  - we model these distinctions by associating different use cases to different actors
- During system design
  - determining which objects are shared among actors, and defining how actors can control access
- In general, we need to define for each actor which operations they can access on each shared object
- We model access on classes with an access matrix
  - rows represent actors
  - columns represent classes whose access we control

Table 7-2: Access Matrix for a Banking System

Rows represents Actors

Columns represents Classes

Table 7-2 Access matrix for a banking system. `Tellers` can perform small transactions and inquire balances. `Managers` can perform larger transactions and access branch statistics in addition to the operations accessible to the `Tellers`. `Analysts` can access statistics for all branches, but cannot perform operations at the account level.

Objects Actors	Corporation	LocalBranch	Account
Teller			postSmallDebit() postSmallCredit() examineBalance()
Manager		examineBranchStats()	postSmallDebit() postSmallCredit() postLargeDebit() postLargeCredit() examineBalance() examineHistory()
Analyst	examineGlobalStats()	examineBranchStats()	

Operations available

Access Control (cont.)

ทำได้ 3 แบบ

- Access matrix can be represented as follows:
  - A **global access table**
    - represent every cell as a (actor, class, operation) tuple
  - An **access control list**
    - associates a list of (actor, operation) pairs with each class to accessed
    - empty cells are discarded
  - A **capability**
    - associates a (class, operation) pair with an actor
    - a capability allows an actor access to an object of the class described in the capability
    - denying a capability is equivalent to denying access

Performance Issue of Access Matrix Represetntation

- The representation of the access matrix is also a performance issue. เลือก rep. ไม่เหมาะจะระบบจะช้า
- Global access tables require a lot of space
- Access control lists make it faster to answer the question, "Who has access to this object?" เช่น class account มีใครมาใช้ได้บ้างและทำอะไรได้บ้าง
- Capability lists make it faster to answer the question, "Which objects has this actor access to?" เช่น นาย ก เข้าถึง class ใดได้บ้างและทำอะไรได้บ้าง

Q&A