CST4125: Blockchain Development
Week: 4
Title: Review and Pizza

Dr Ian Mitchell

smerf.net
Bedfordshire,
UK

October 2023

Staff Etiquette

Academics

- Record
  - Chatrooms
  - Live
  - Attendance
- Mute control
- Access control
- No anonymity
- Share personal information via screen shares

Copyright

Student Etiquette

Do's

- Behave as normal, be respectful
- No anonymity
- First and last names to identify you
- Kindness/Difficulty
- Be patient, some one may have technical issues
- Mute microphone, unless speaking
- Use chatroom appropriately
- Keep video on, especially when talking
- Tolerance

Don'ts

- Share personal information
- Try not to multi-task
- Behave inappropriately
- Bully other students
- Disruption
- No eating

Labs

- Complete exercise together
- All leave room
- Try exercise
- Have questions or queries
- Enter waiting room for 1-2-1

Deadlines

				Feedback	
Description	Submission	Weight	Deadline	Formative	Summative
1. Hyperledger	MyLearning	50%	14th April 2023	LW11-12	10/05/2023
2. Ethereum	MyLearning	50%	8th July 2023	LW23-24	28/07/2023
Resits	MyLearning	50-100%	14th August 2023	None	None
Deferals	MyLearning	50-100%	14th August 2023	None	None

Coursework 1

- **Problem Definition: 14%**
- **Data Modelling: 18%**
- **Access Control Language: 16%**
- Business Logic: 26
- Documentation: 4%
- Presentation: 20%
- Business Network Archive: 2%
- **Deadline: 14th April 2023**

Problem Definition: 14%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
Problem Definition, PD (14%)	Specification	No Spec.	Spec., present	Spec. is not conducive to BC	Unrelated or missing spec. components	Spec. conducive to BC, all components explained and coherent	1	/4
	Flowchart, FC	No use of FC in [5]	FC applied, no explanation.	All components of FC applied, some explanation.	All components of FC applied correctly but does not match spec/UCD.	All components of FC applied correctly and matches spec/UCD	1	/4
	Use Case Diagram, UCD.	No UCD	Incoherent UCD	Misaligned UCD and PD. Assumptions left uncommented	No include or extend relationships. Assumptions commented	Aligned and complete UCD with comments and assumptions	1	/4

Data Modelling: 18%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
Data Model (18%)	Participants	No participants	Lacking and/or incorrect participants. Incorrect data types used. Unidentified.	Irrelevant participants. Correct identification. Lacking any assumptions. Opportunities to use more appropriate data types missed.	Participants lacking UCs and incomplete assumptions. Structurally sound.	Correct participants, data structures, assumptions and matching UCs	1	/4
	Assets	No assets	Lacking and/or incorrect assets	Irrelevant assets. No enum or concepts.	Assets unrelated to participants or no assets with the capability of state change	Some of the assets must at least be 3 of the following: have a state capable of change, relevant, complete and related to participants	1	/4
	Transactions, TX	No TX	Vague TX	TX not updating state	TX without ownership	participant specific TX	1	/4
	Comments	No comments	Auto-generated comments only (headers only), no clarifying comments	Vague, incorrectly placed and/or unexplanatory comments	Explanatory and identifiable comments, but incomplete. Too verbose and high comment to code ratio	Complete, concise and succinct comments	1	/4

Access Control Language: 16%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
Access Control Language, ACL, (16%)	Participants	No ACL. Basic ACL, admin access only & automatically generated code	ACL has too few rules	ACL has contradictions or allows unauthorised access to transactions or assets. There is no difference between participant access	ACL order is incorrect	ACL is implemented correctly	1	/4
	Ordering, Comments and listing	No listing or basic ACL, admin access only & automatically generated code	Syntax errors for ACL.	Rules are disorganised and need re-ordering. Inclusion of commented out rules	Rules are in correct order, but lack ideal names, descriptor values and comments. No line numbers.	Correct order and appropriate names, descriptors values and comments	1	/4
	Conditions	Auto-generated rules only. Admin access to all.	No conditions and simple rules only	Conditions applied incorrectly.	Identifier conditions applied correctly	Conditions to check status or lists and of a higher order of difficulty	1	/4

Business Logic: 26%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
Business Logic (26%)	Queries	No Queries	Queries but don't execute	Irrelevant Queries	Relevant Queries without relationships	Relevant Queries with relationships	1	/4
	Transactions	No Transactions	BL - run time execution	BL code accessing assets and participants, with no restriction, or comments directing to ACL	BL code accessing TX with restrictions, but not acknowledged	Acknowledged rules and restrictions and code accessing both assets and participants correctly	2	/8
	API	No use of promises	BL code not executing	BL code duplicating ACL	No extensive use of API and promises	Extensive use of API and Promises and complexity used to aid the update of state correctly	3	/12
	Initialise	No initialisation or automatic population of values in registry	Initialisation present but not working	Initialisation only partial, e.g., only completes assets and not participants	All assets and participants populated but incorrectly, e.g., data is misaligned	All assets and participants populated correctly	1	/4
	Comments	No comments	Non-explanatory comments	Partial explanatory comments	Overly commented	Fully explanatory comments	1	/4

Presentation: 20%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
Presentation (20%)	Slide Content	No slides	Incoherent presentation and not demonstrating the understanding of the coursework. Cluttered and/or illegible slide content	Coherent but poor content coverage. Less than 5 mins in length. Uncluttered. Some Illegible slide content, especially screenshots	Less than 9 mins or greater than 10 mins. Clear figures and screenshots. Coherent but not explaining all points required	Between 9-10 mins in length, clear and readable slides and addresses all items	1	/4
	Transaction, TX	No Demonstration	Demonstration of successful TX	Demonstration of unsuccessful TX due to ACL	Demonstration of unsuccessful Demonstration due to BL	All demonstrations completed	1	/4
	Structure	No structure	No headers *and* footers, slide numbers	No headers *or* footers, slide numbers	Headers, Footers and numbers but incorrect	All slides consistent with correct information in headers and footers.	1	/4

Documentation: 4%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
Report (4%)	English	Many sentences rendered nonsensical and many misspellings	Some sentences rendered nonsensical and a few misspellings	Sentences with poor grammar, written in first or second person, and a few misspellings	Good grammar, not written in third person. A few grammatical and spelling mistakes.	Written in third person. A few grammatical or spelling mistakes	1	/4
	Template	No structure followed	No numbering but structure present	Incorrect frontmatter or backmatter, but main matter correct structure. No figure, listing or table captions.	No citations or references, or incorrect bibliography style applied	Correct template, citations/references, numbering and template compliance.	1	/4

Business Network Archive: 2%

Criteria	Sub-criteria	0	1	2	3	4	W	Σ
BNA (2%)	Execution	Errors	Run-time errors	No errors (4)			1	/4
	BNA format	None	ACL	Node.js	CTO	Structure	1	/4

Lecture Aims

Aims

The aims are to provide you with formative feedback by completing a blockchain application for a pizza delivery restaurant.

Disclaimer

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

Lecture Objectives

Knowledge

- Participants
- Assets
- Transactions
- Concepts
- Enumerators
- ACL
- Use Case Diagrams

Use Cases

- Jacobson [2]
- UML - Unified Modelling Language
- Describe behaviour of system
- Define the system's boundary
- No data structure, algorithms, etc
- Maps user to functions
- High level of abstraction

- System - What is being described
- Actors - Who interacts with the system
- Use Cases - what the actors can do

UML Reading

UML@ classroom: An introduction to object-oriented modeling,
Seidl, Martina and Scholz, Marion and Huemer, Christian and Kappel, Gerti
2015
Springer

Actors

- Roles
- Trigger the Use Case
- Interacts with the system
- One person can have many roles
- Name describes the role, not the person

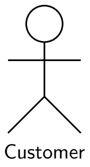


Figure: An actor named Customer

Actors

- Roles
- Trigger the Use Case
- Interacts with the system
- One person can have many roles
- Name describes the role, not the person
- **Participants**

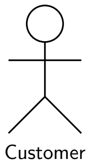


Figure: An actor named Customer

Use Case

- Function
- Invoked by a trigger
  - Environment
  - Actor
  - Another Use Case
- Ellipse

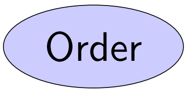


Figure: Use Case: Order

Use Case

- Function
- Invoked by a trigger
  - Environment
  - Actor
  - Another Use Case
- Ellipse
- **Transactions**

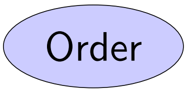
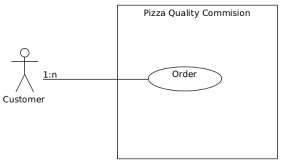


Figure: Use Case: Order

Associations

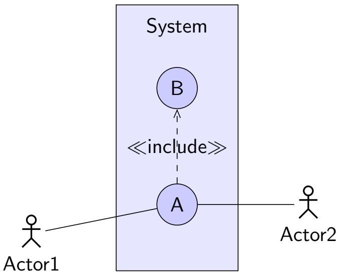
- connection between:
  - Actors and Use Cases
  - Use Cases
- Multiplicity
  - rarely specified
- User adopts role of actor
- This role is authorise to execute use case



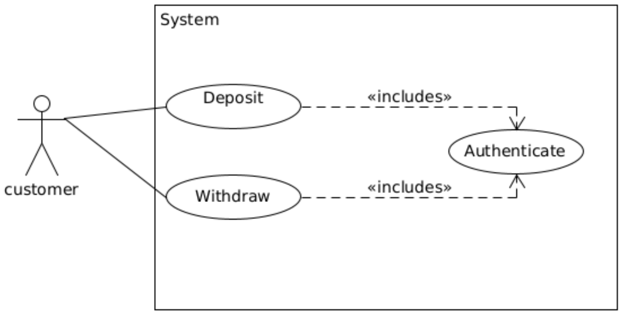
Relationships between Use Cases

Include

- Includes
- instance of A includes instance of B
- When an A is invoked it includes B
- used to break down complex use cases
- A: Base use case
- B: included use case
- The base use case always the behaviour of the included use case



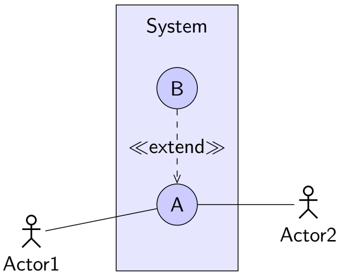
Include Example



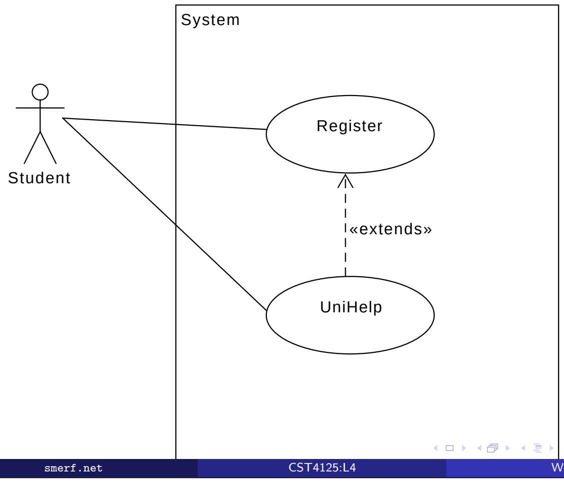
Relationships between Use Cases

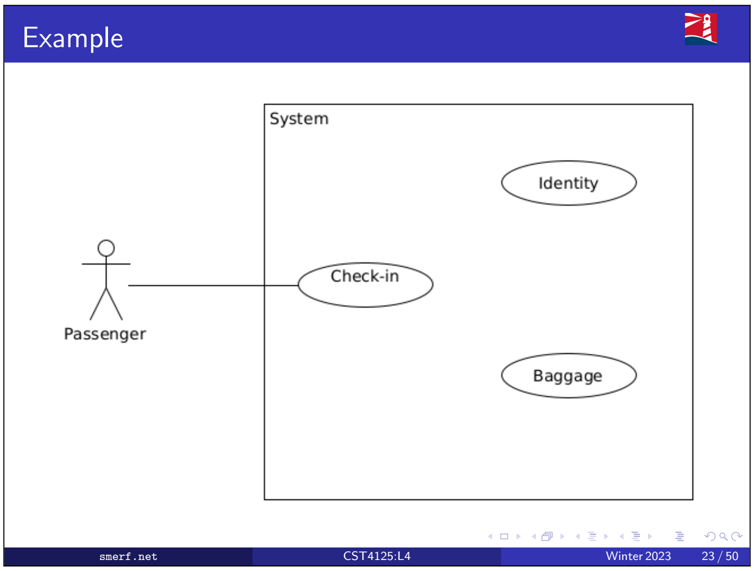
Extends

- Extends
- A *may* use the behaviour of B
- can insert the behaviour of B in A
- A: base use case
- B: extending use case
- A and B can be executed independently



Include Example





Example

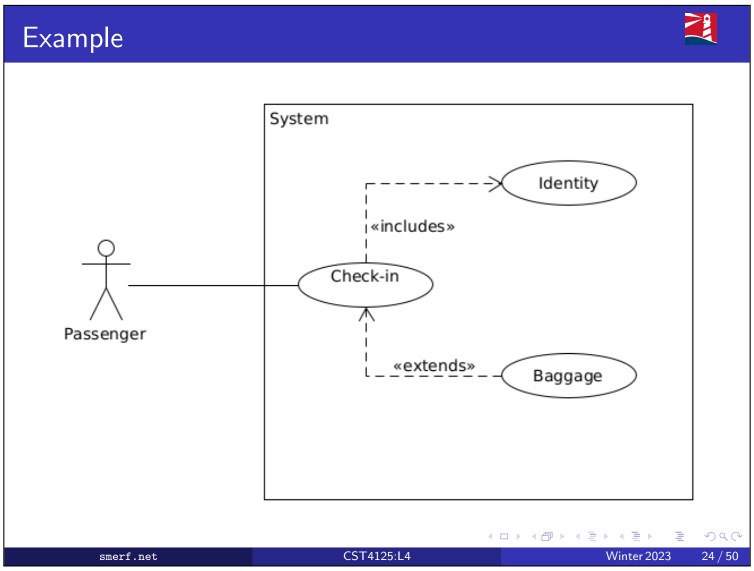
System

Identity

Check-in

Baggage

Passenger



Example

System

Identity

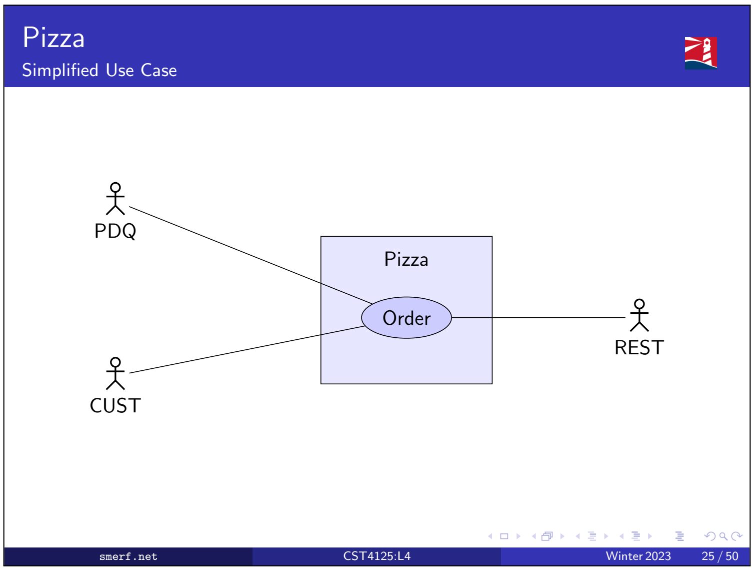
«includes»

Check-in

«extends»

Baggage

Passenger



Pizza

Simplified Use Case

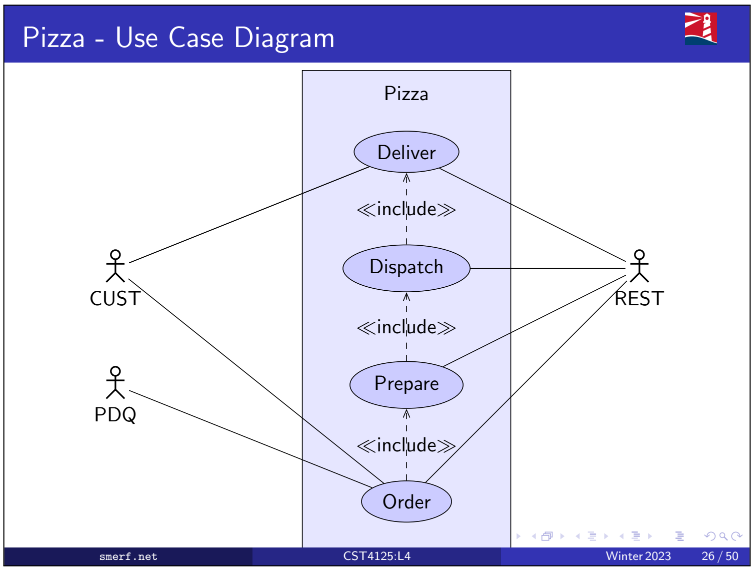
PDQ

Pizza

Order

REST

CUST



Pizza - Use Case Diagram

Pizza

Deliver

≪include≫

Dispatch

≪include≫

Prepare

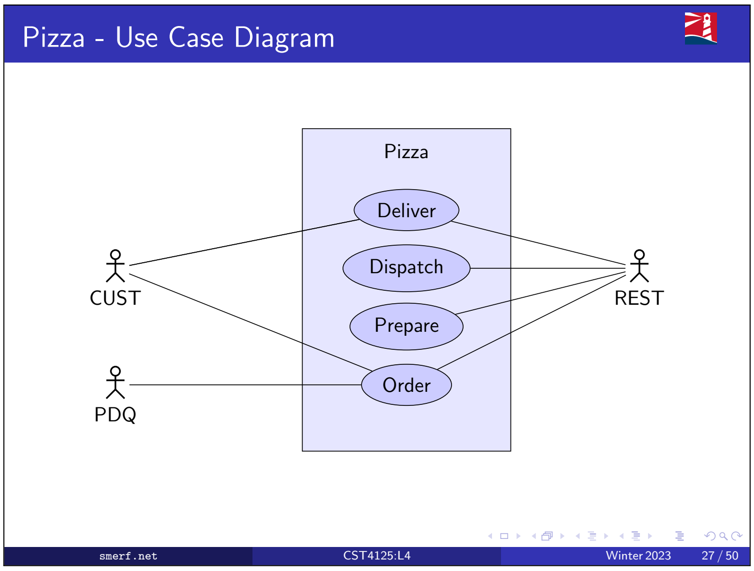
≪include≫

Order

CUST

PDQ

REST



Pizza - Use Case Diagram

Pizza

Deliver

Dispatch

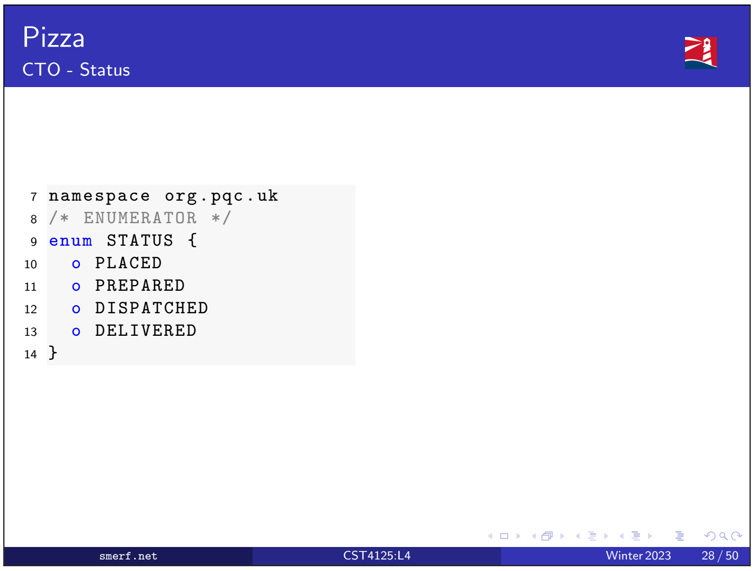
Prepare

Order

CUST

PDQ

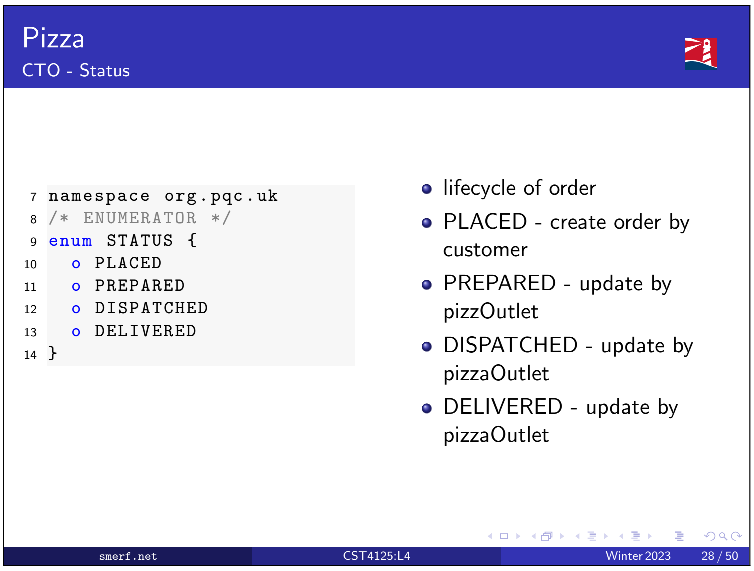
REST



Pizza

CTO - Status

```
namespace org.pqc.uk
/* ENUMERATOR */
enum STATUS {
   o PLACED
   o PREPARED
   o DISPATCHED
   o DELIVERED
}
```

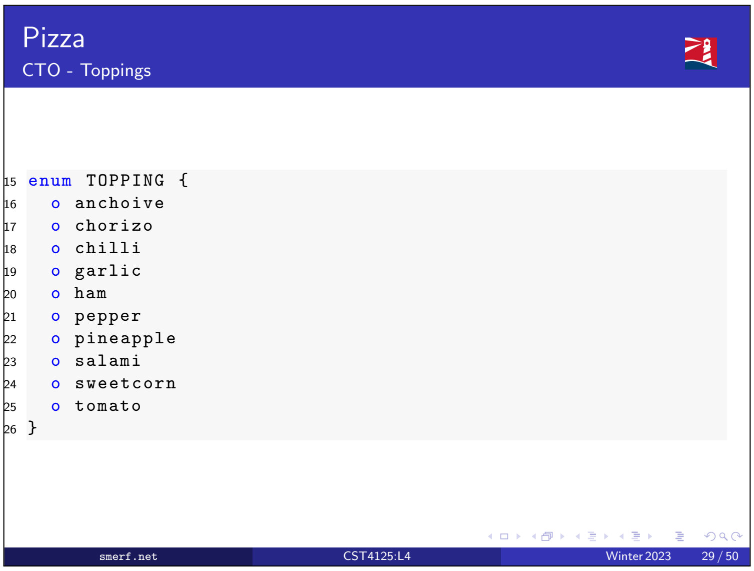


Pizza

CTO - Status

```
namespace org.pqc.uk
/* ENUMERATOR */
enum STATUS {
  o PLACED
  o PREPARED
  o DISPATCHED
  o DELIVERED
}
```

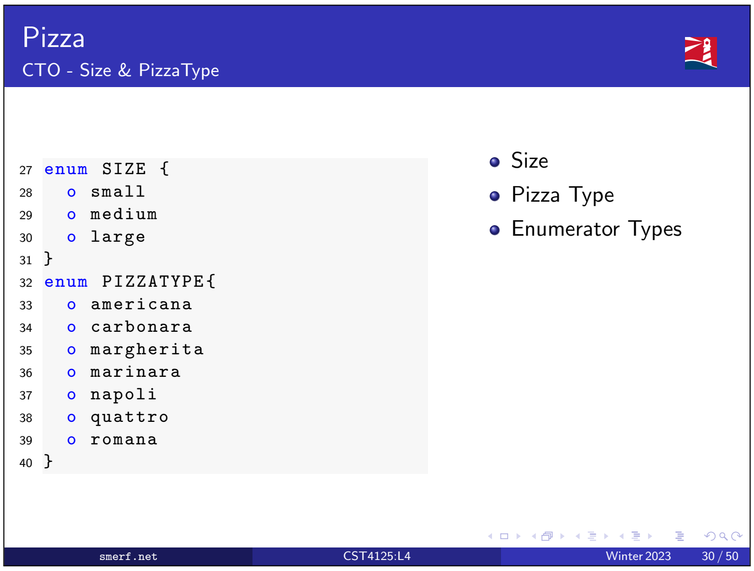
- lifecycle of order
- PLACED - create order by customer
- PREPARED - update by pizzOutlet
- DISPATCHED - update by pizzaOutlet
- DELIVERED - update by pizzaOutlet



Pizza

CTO - Toppings

```
enum TOPPING {
  o anchoive
  o chorizo
  o chilli
  o garlic
  o ham
  o pepper
  o pineapple
  o salami
  o sweetcorn
  o tomato
}
```

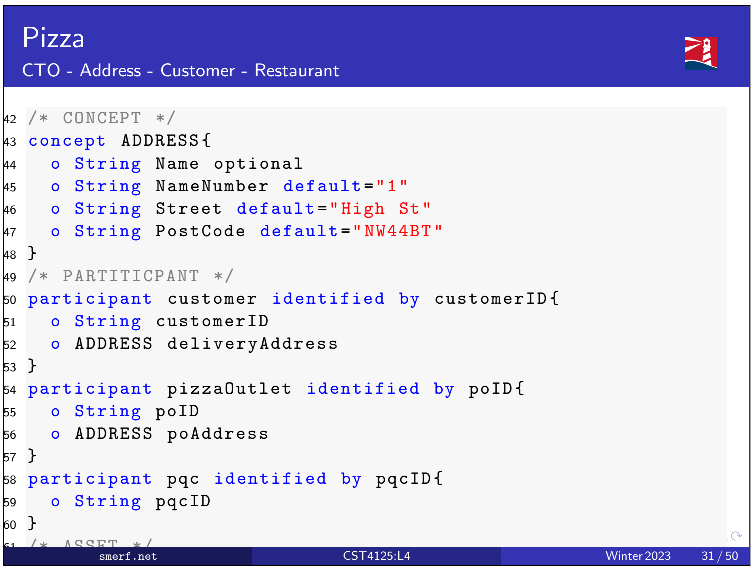


Pizza

CTO - Size & PizzaType

```
enum SIZE {
  o small
  o medium
  o large
}
enum PIZZATYPE{
  o americana
  o carbonara
  o margherita
  o marinara
  o napoli
  o quattro
  o romana
}
```

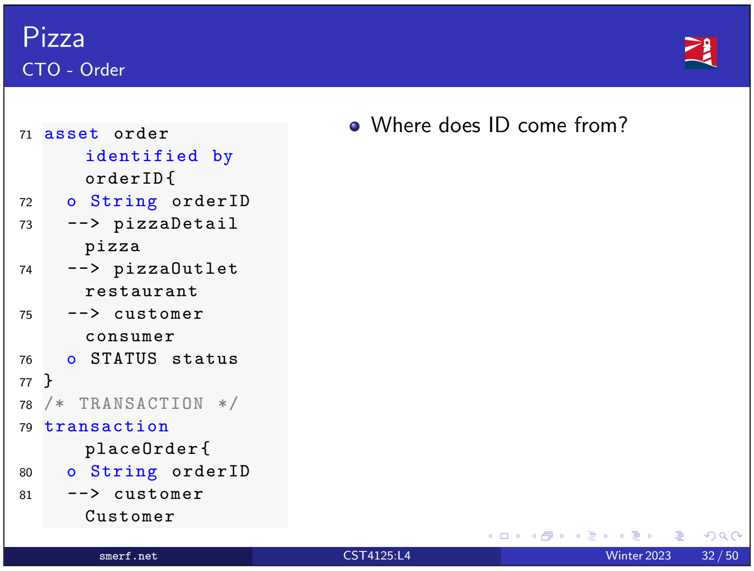
- Size
- Pizza Type
- Enumerator Types



Pizza

CTO - Address - Customer - Restaurant

```
/* CONCEPT */
concept ADDRESS{
  o String Name optional
  o String NameNumber default="1"
  o String Street default="High St"
  o String PostCode default="NW44BT"
}
/* PARTITICPANT */
participant customer identified by customerID{
  o String customerID
  o ADDRESS deliveryAddress
}
participant pizzaOutlet identified by poID{
  o String poID
  o ADDRESS poAddress
}
participant pqc identified by pqcID{
  o String pqcID
}
/* ASSET */
```

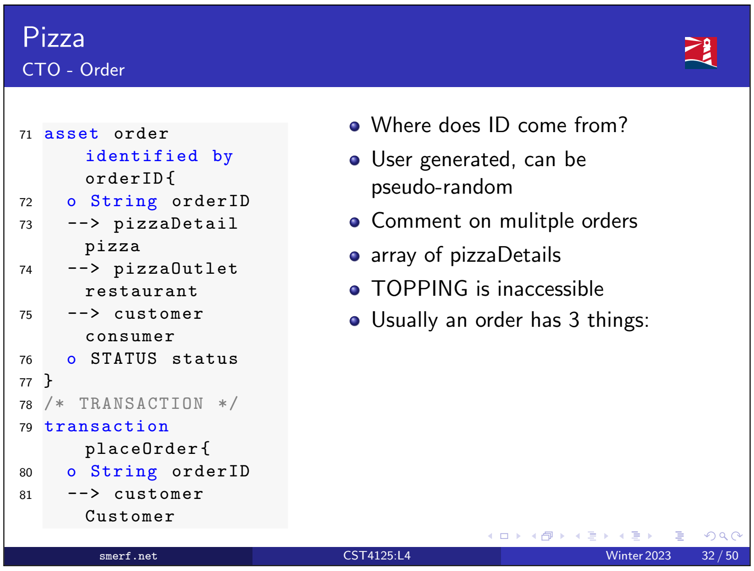


Pizza

CTO - Order

```
asset order
    identified by
    orderID{
  o String orderID
  --> pizzaDetail
    pizza
  --> pizzaOutlet
    restaurant
  --> customer
    consumer
  o STATUS status
}
/* TRANSACTION */
transaction
    placeOrder{
  o String orderID
  --> customer
    Customer
```

- Where does ID come from?



Pizza

CTO - Order

```
asset order
    identified by
    orderID{
  o String orderID
  --> pizzaDetail
    pizza
  --> pizzaOutlet
    restaurant
  --> customer
    consumer
  o STATUS status
}
/* TRANSACTION */
transaction
    placeOrder{
  o String orderID
  --> customer
    Customer
```

- Where does ID come from?
- User generated, can be pseudo-random
- Comment on mulitple orders
- array of pizzaDetails
- TOPPING is inaccessible
- Usually an order has 3 things:

Pizza

CTO - Order

```
asset order
    identified by
    orderID{
  o String orderID
  --> pizzaDetail
    pizza
  --> pizzaOutlet
    restaurant
  --> customer
    consumer
  o STATUS status
}
/* TRANSACTION */
transaction
    placeOrder{
  o String orderID
  --> customer
    Customer
```

- Where does ID come from?
- User generated, can be pseudo-random
- Comment on mulitple orders
- array of pizzaDetails
- TOPPING is inaccessible
- Usually an order has 3 things:
  1. Product: Pizza, sometimes the quantity
  2. Seller: Restaurant
  3. Buyer: Customer
- STATUS: track progress

Pizza Enums

State

```
/* ENUMERATOR */
enum STATUS {
  o PLACED
  o PREPARED
  o DISPATCHED
  o DELIVERED
}
```

Pizza Enums

```
enum TOPPING {
  o anchoive
  o chorizo
  o chilli
  o garlic
  o ham
  o pepper
  o pineapple
  o salami
  o sweetcorn
  o tomato
}
```

Pizza Enums

```
enum SIZE {
  o small
  o medium
  o large
}
enum PIZZATYPE{
  o americana
  o carbonara
  o margherita
  o marinara
  o napoli
  o quattro
  o romana
}
```

Pizza Concepts

```
/* CONCEPT */
concept ADDRESS{
  o String Name optional
  o String NameNumber default="1"
  o String Street default="High St"
  o String PostCode default="NW44BT"
}
```

Pizza Participants

```
/* PARTITICPANT */
participant customer identified by customerID{
  o String customerID
  o ADDRESS deliveryAddress
}
participant pizzaOutlet identified by poID{
  o String poID
  o ADDRESS poAddress
}
participant pqc identified by pqcID{
  o String pqcID
}
```

Pizza Asset

```
/* ASSET */
asset pizzaDetail identified by pID{
  o String pID
  o PIZZATYPE pizzaType
  o SIZE pizzaSize
  o TOPPING[] extras optional
}
```

Pizza Asset

```
asset order identified by orderID{
  o String orderID
  --> pizzaDetail pizza
  --> pizzaOutlet restaurant
  --> customer consumer
  o STATUS status
}
```

Pizza

CTO - Transactions

```
/* TRANSACTION */
transaction placeOrder{
  o String orderID
  --> customer Customer
  --> pizzaDetail pizza
  --> pizzaOutlet restaurant
}
transaction prepareOrder{
  --> order pizzaPrepared
}
transaction dispatchOrder{
  --> order pizzaDispatched
}
transaction deliverOrder{
  --> order pizzaDelivered
}
```

Pizza Events

```
/* EVENTS */
event dispatchOrderEvent{
  --> order pizzaDispatched
}
event prepareOrderEvent{
  --> order pizzaPrepared
}
event deliverOrderEvent{
  --> order pizzaDelivered
}
```

CTO review

- State
  - Enumerator
  - Asset Status
  - Transaction per state
- Transactions
  - minimise TX
  - 1 order per tx
  - 1 order could be many pizza's

Rules

ACL - Customer

```
rule customerSeeSelf{
  description: "customer see
    themselves"
  participant(p): "org.pqc.uk.
    customer"
  operation: ALL
  resource(r): "org.pqc.uk.
    customer"
  condition: (p.getIdentifier
    ()==r.getIdentifier())
  action: ALLOW
}
rule customerSeePizza{
  description: "customer see
    pizza"
  participant: "org.pqc.uk.
    customer"
  operation:READ
  resource: "org.pqc.uk.
```

CustomerSeeSelf:
: Customers can only see themselves. Condition that ensures the consumer in the order is equal to the customer.

CustomerSeePizza:
: Customers can see the pizzas available

Rules

ACL - Customer

```
rule customerPlaceOrder{
  description: "customer places order"
  participant: "org.pqc.uk.customer"
  operation: ALL
  resource: "org.pqc.uk.placeOrder"
  action: ALLOW
}
rule customerReadRestaurant{
  description: "customer has read access to restaurants"
  participant: "org.pqc.uk.customer"
  operation: READ
  resource: "org.pqc.uk.pizzaOutlet"
```

customerPlaceOrder:

Only a customer can place an order and access transaction `placeOrder`

customerReadRestaurant:

Customers are permitted to read pizzaOutlet details

Rules

ACL - Restaurant

```
rule restaurantSeeSelf{
  description: "restaurants can only view their own details"
  participant(p): "org.pqc.uk.pizzaOutlet"
  operation: ALL
  resource(r): "org.pqc.uk.pizzaOutlet"
  condition: (p.getIdentifier()==r.getIdentifier())
  action: ALLOW
}
rule restaurantSeeOrders{
  description: "restaurant can only see their own orders"
  participant(p): "org.pqc.uk.pizzaOutlet"
```

restaurantSeeSelf:

Restaurant can only see themselves

restaurantSeeOrders:

Restaurant can only see orders placed at their pizzaOutlet

Rules

ACL - Restaurant

```
rule restaurantReadsCustomer{
  description: "restaurant reads customer"
  participant: "org.pqc.uk.pizzaOutlet"
  operation: READ
  resource: "org.pqc.uk.customer"
  action:ALLOW
}
rule restaurantPlaceOrder{
  description: "restaurant reads order"
  participant: "org.pqc.uk.pizzaOutlet"
  operation: READ, UPDATE// CANNOT CREATE
  resource: "org.pqc.uk.order"
  transaction: "org.pqc.uk.
```

restaurantReadsCustomer:

restaurant can read customer details

restaurantPlaceOrder:

Restaurants cannot place orders, merely read and update the status of them

restaurantProcessOrder:

Restaurants can process orders from status `PLACED` to `PREPARED` using transaction `prepareOrder`

Rules

ACL - Restaurant

```
rule restaurantDispatchOrder{
  description: "restaurant dispatch order access"
  participant: "org.pqc.uk.pizzaOutlet"
  operation:ALL
  resource:"org.pqc.uk.dispatchOrder"
  action:ALLOW
}
rule restuarantDeliverOrder{
  description: "restaurant deliver order access"
```

restaurantDispatchOrder:

Restaurant can process orders from status `PREPARED` to `DISPATCHED` using transaction `restaurantDispatchOrder`

restaurantDeliverOrder:

Restaurant can process orders from status `DISPATCHED` to `DELIVERED` using the transaction `restaurantDeliverOrder`

Ubuntu Install UMLet

- `sudo apt update -y`
- `sudo apt-get install umlet -y`
- Reading:
  - Chapter 3 in [3]
  - Misuse Case [4]
- Online: umlet

References

[1] Nitin Gaur et al. *Hands-on Blockchain with Hyperledger: Building Decentralised Applications with Hyperledger Fabric and Composer.* Packt, 2018. ISBN: 9781788994521.

[2] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach.* Addison-Wesley, 1992.

[3] Martina Seidl et al. *UML@ classroom: An introduction to object-oriented modeling.* Springer, 2015.

[4] Guttorm Sindre and Andreas L Opdahl. “Eliciting security requirements with misuse cases”. In: *Requirements engineering* 10.1 (2005), pp. 34–44.

[5] Dylan Yaga et al. *Blockchain technology overview.* Tech. rep. National Institute of Standards and Technology, 2018.

Web Resources

- `http://hyperledger.org`