Software Development with UML

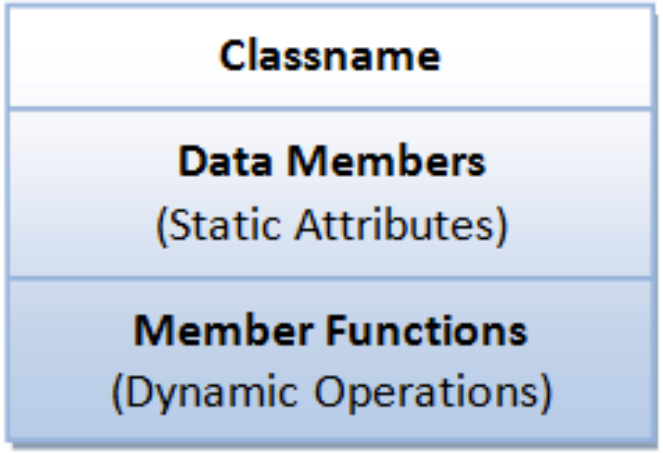
Software Development for Robotics and Automation Engineering

Classes & Instances

- **Class**: A class is a definition of objects of the same kind. In other words, a class is a blueprint, template, or prototype that defines and describes the static attributes and dynamic behaviors common to all objects of the same kind.
- **Instance / Object**: An instance is a realization of a particular item of a class. In other words, an instance is a copy of the class with actual values. All the instances of a class have similar properties, as described in the class definition. It represents an identifiable entity in the real world. For example, you can define a class called "Student" and create three instances of the class "Student" for "Peter", "Paul" and "Pauline".
- The term "**object**" usually refers to instance. But it is often used quite loosely, which may refer to a class or an instance.

https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

Class



A class is a 3-compartment box encapsulating data and functions

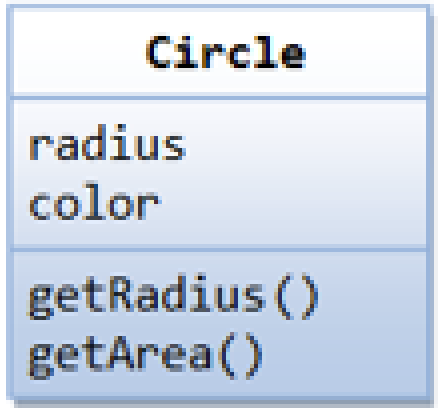
1. Classname (or identifier): identifies the class.
2. Data Members or Variables (or attributes, states, fields): contains the static attributes of the class.
3. Member Functions (or methods, behaviors, operations): contains the dynamic operations of the class.

In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.

Class Members: The data members and member functions are collectively called class members.

https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

Class Definition



C++

```
class Circle {           // classname
private:
     double radius;      // Data members (variables)
     string color;
public:
     Circle(float radius, int color);
     double  getRadius();// Member functions
     double  getArea();
}
```

Python

```
import math

class Circle:
  # class attribute
  count = 0
  # initializer / instance attributes
  def __init__(self, radius=0, color=None):
    self.radius = radius
    self.color = color
    Circle.count += 1

  # instance method
  def getRadius(self):
    return self.radius

  # instance method
  def getArea(self):
    return math.pi*self.radius*self.radius
```

https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

Creating Instances of a Class

To create an instance of a class, you have to:

1. Declare an instance identifier (name) of a particular class.
2. Invoke a constructor to construct the instance (i.e., allocate storage for the instance and initialize the variables).

```
# Construct 3 instances of the class Circle: c1, c2, and c3
c1 = Circle(4, "red")       # radius, color
c2 = Circle(3.4)            # radius, default color
c3 = Circle()               # default radius and color
```

https://www.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

UML : Unified Modeling Language

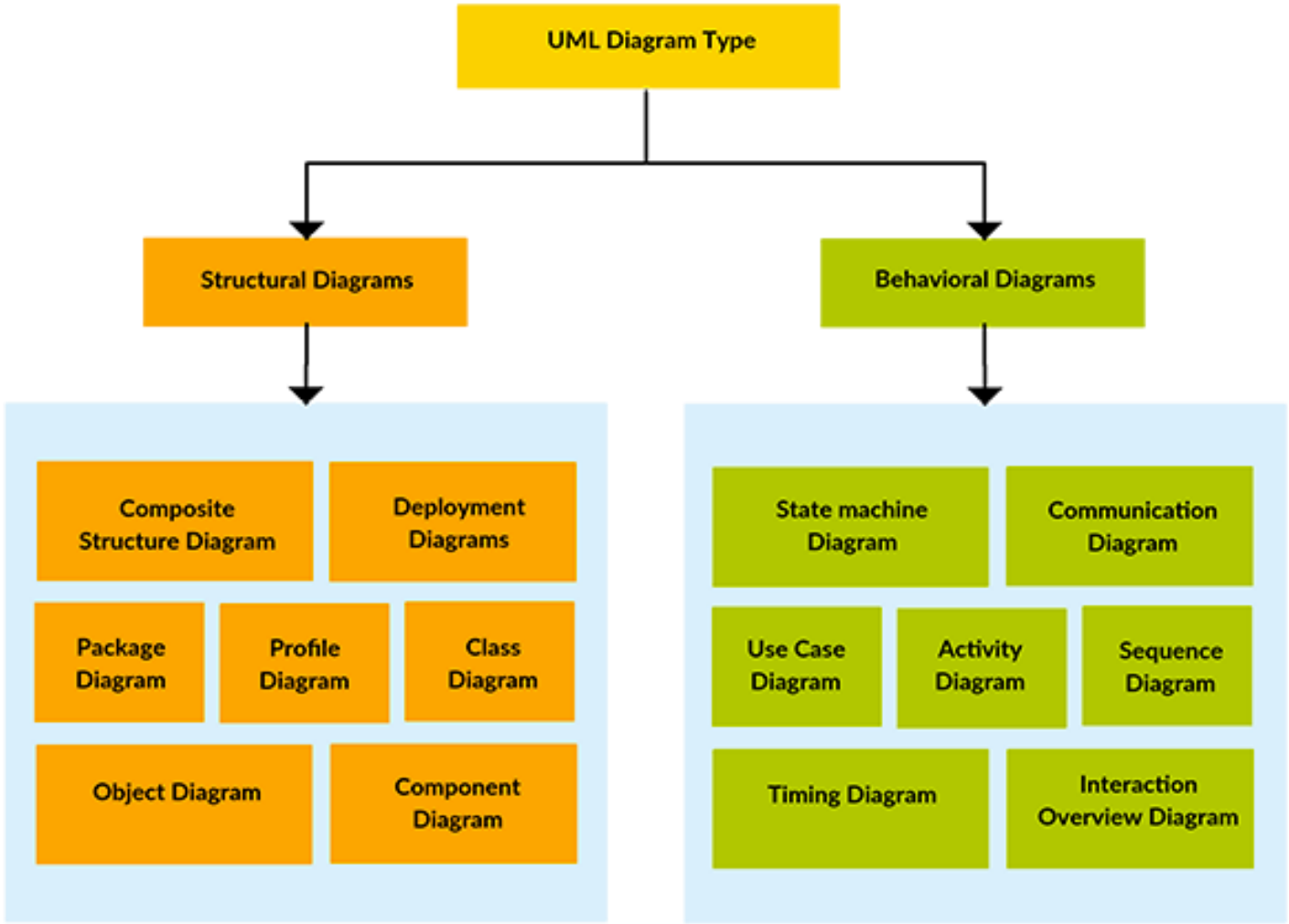
- UML is used in object oriented software engineering. Although typically used in software engineering it is a rich language that can be used to model an application structures, behavior and even business processes.
- There are **14 UML diagram types** to help you model these behavior.

Class Diagram
Component Diagram
Deployment Diagram
Object Diagram
Package Diagram
Profile Diagram
Composite Structure Diagram
Use Case Diagram
Activity Diagram
State Machine Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

Types of UML diagram

- **Structure diagrams** show the things in a system being modeled. In a more technical term, they show different objects in a system.

- **Behavioral diagrams** shows what should happen in a system. They describe how the objects interact with each other to create a functioning system.



http://creately.com/blog/diagrams/uml-diagram-typ

UML Core Conventions

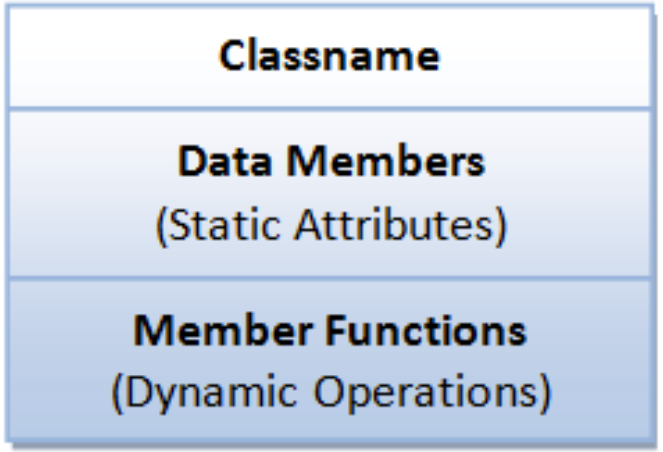
- Rectangles are classes, instances (objects) or states
- Ovals are functions or use cases
- Instances are denoted with an underlined names (object:class)
  - `myWatch:SimpleWatch`
  - `Joe:Firefighter`
- Types are denoted with non underlined names
  - `SimpleWatch`
  - `Firefighter`
- Diagrams are graphs
  - Nodes are entities
  - Arcs are relationships between entities

Class Diagram

- Class diagrams are arguably the most used UML diagram type. It is the main building block of any object oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between each class.

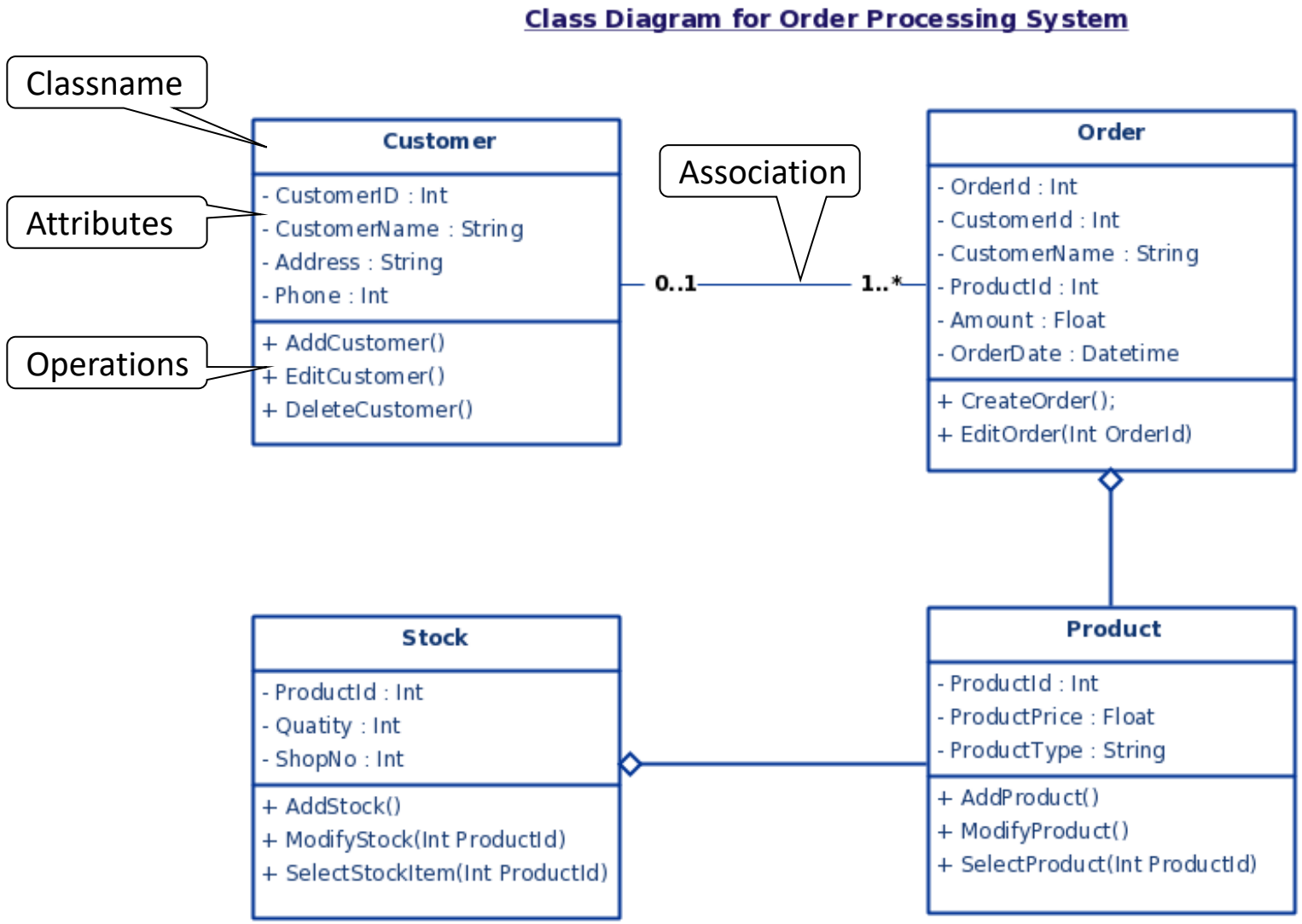
- In most modeling tools, a class has three parts, Classname at the top, attributes in the middle and operations or methods at the bottom.

- In large systems with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows.



A class is a 3-compartment box encapsulating data and functions

Class Diagram Example



A ***class*** represent a concept
A class encapsulates state ***(attributes)*** and behavior ***(operations).***

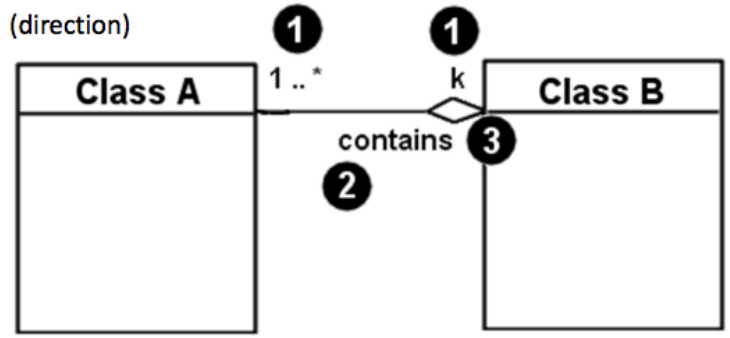
Class diagrams represent the structure of the system.
Used

- during requirements analysis to model problem domain concepts
- during system design to model subsystems and interfaces
- during object design to model classes.

Visibility
+ : public
- : private
: protected

http://creately.com/blog/diagrams/uml-diagram-types-examples/

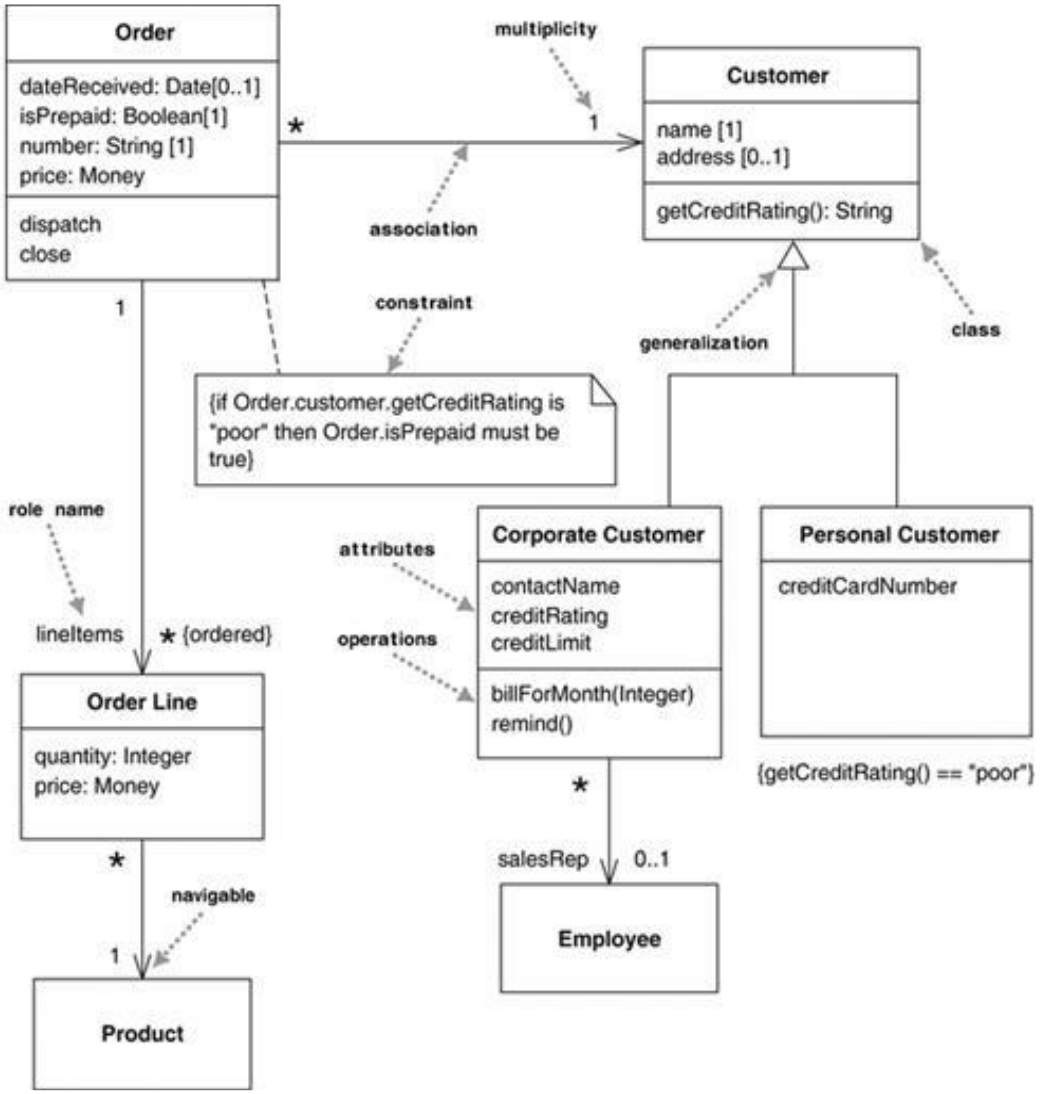
Associational Relationships



1. multiplicity (how many are used)

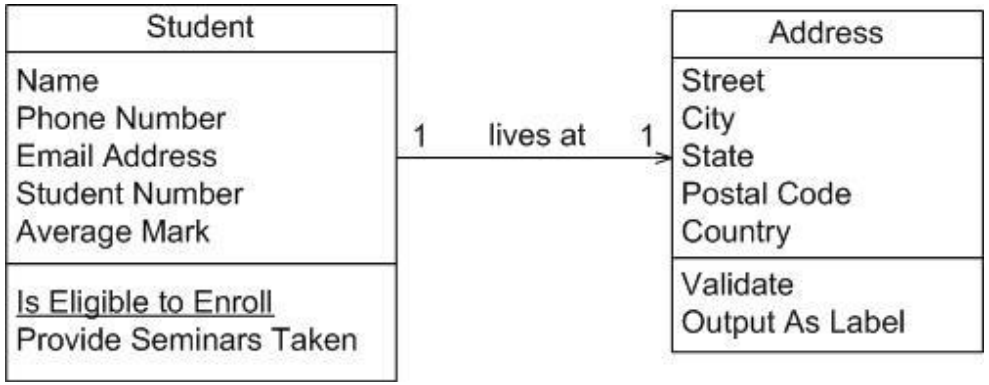
Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

2. name (what relationship the objects have)
3. navigability (direction)

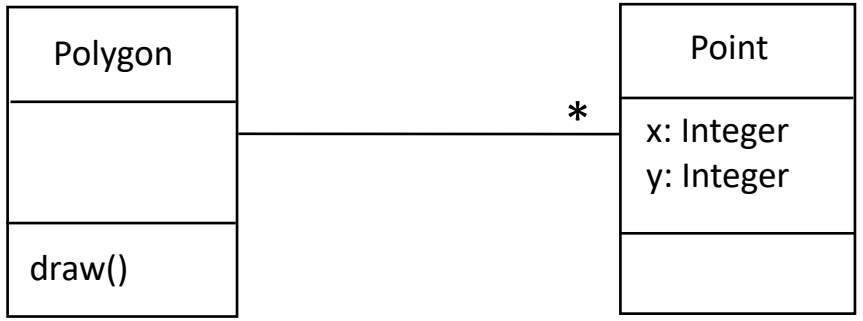


Ref: UML Distilled 3rd Ed

Example of Associations



One-to-one association



One-to-many association

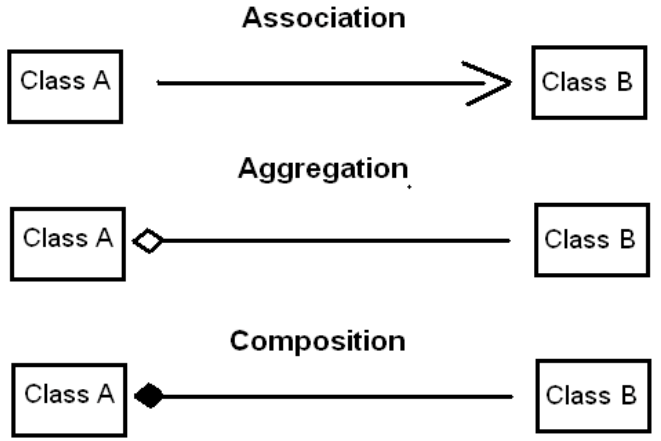
Many-to-many association

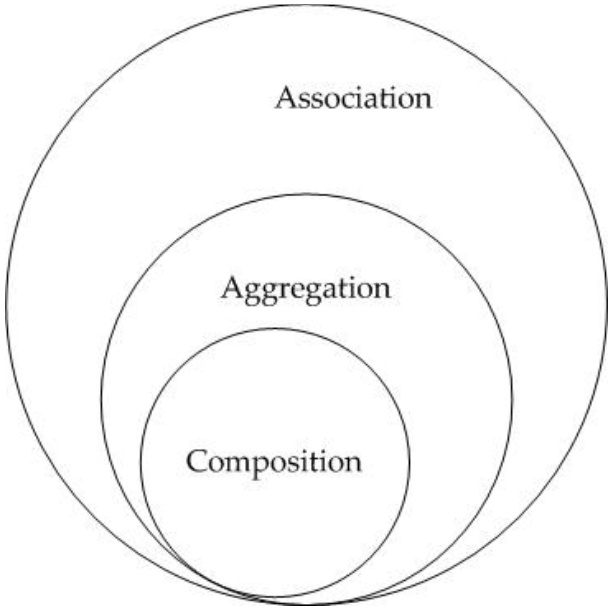
Associational Relationships

Association is a relationship where all objects have their own lifecycle and there is no owner

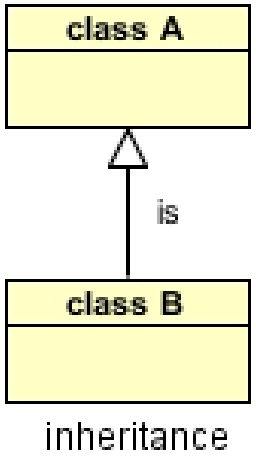
Aggregation is a specialized form of Association where all objects have their own lifecycle, but there is ownership and child objects can not belong to another parent object.

Composition is again specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object does not have its lifecycle and if parent object is deleted, all child objects will also be deleted.





Inheritance lets us create new classes from existing classes. The new classes that we create from the existing classes are called the **derived classes**; the existing classes are called the **base classes**. The derived classes inherit the properties of the base classes.



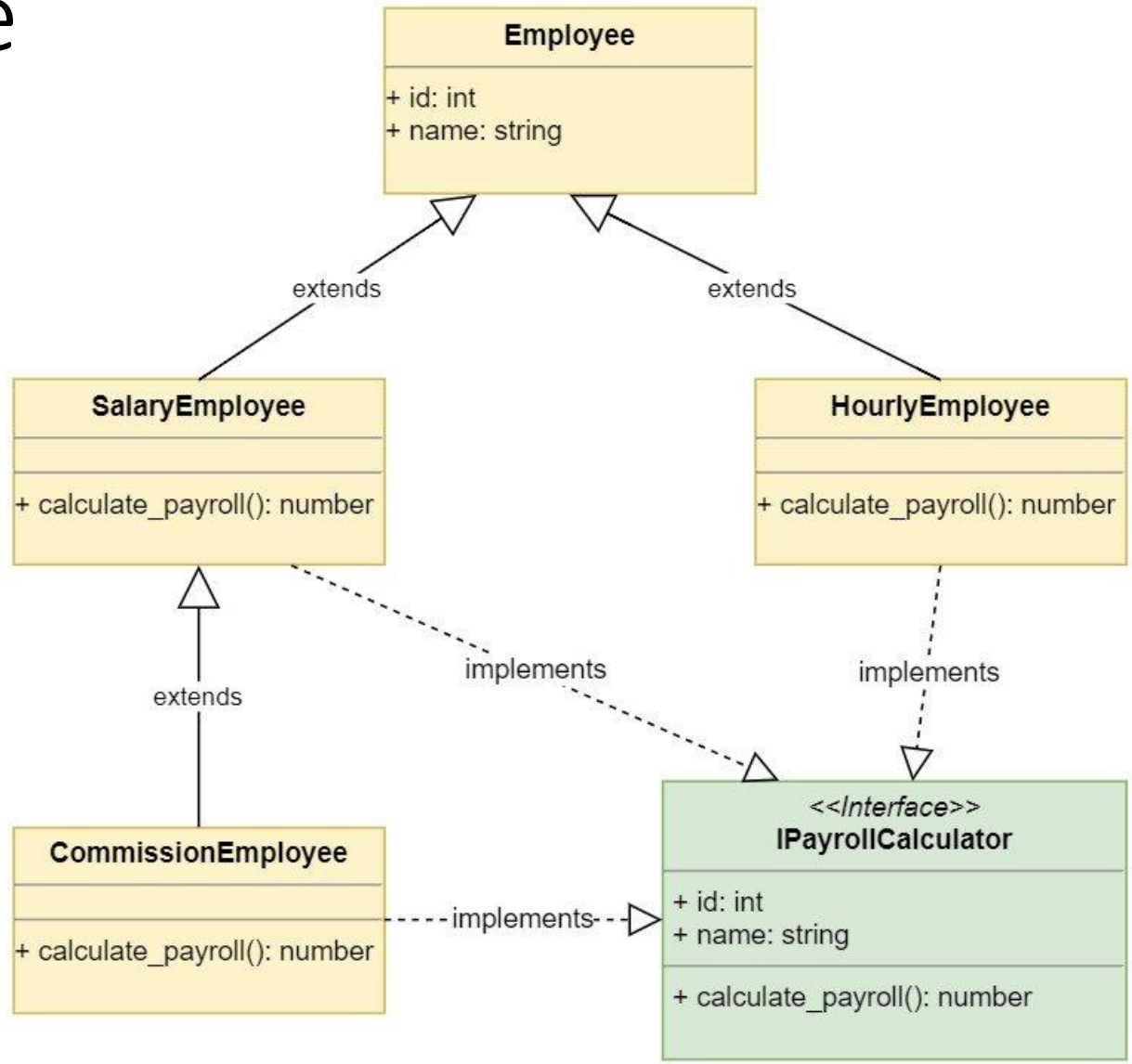
https://stackoverflow.com/questions/885937/what-is-the-difference-between-association-aggregation-and-composition

Inheritance Example

Generalization relationships denote inheritance between classes.

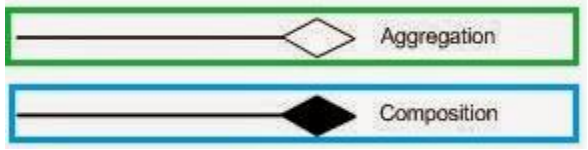
The children classes inherit the attributes and operations of the parent class.

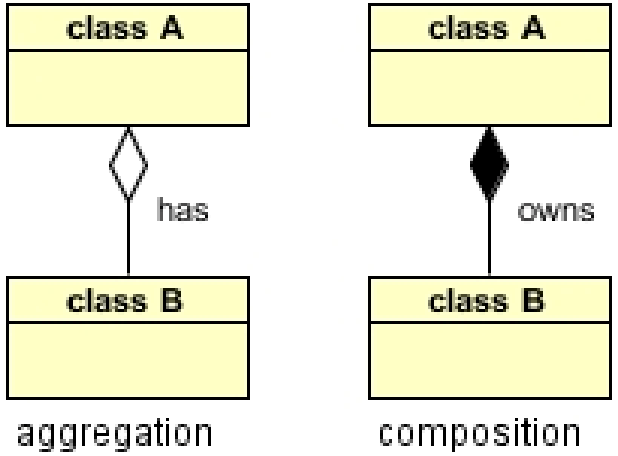
Inheritance simplifies the model by eliminating redundancy.



https://realpython.com/inheritance-composition-python/

Composition vs Aggregation



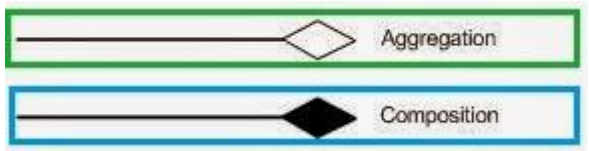


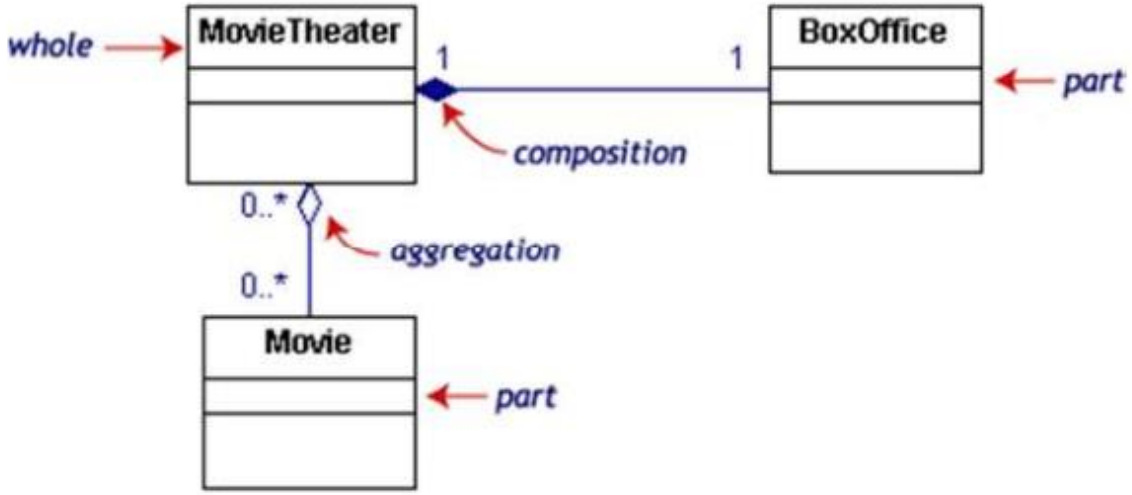
A "owns" B = **Composition** : B has no meaning or purpose in the system without A

A "uses" B = **Aggregation** : B exists independently (conceptually) from A

- Aggregation is a simple collection, like a bag of marbles.
- Composition implies internal/functional dependencies, like the hinges on a box.
- Cars aggregate passengers; they get in and out without breaking the car's functionality.
- The tires are components; remove one and the car no longer functions correctly.

Composition vs Aggregation

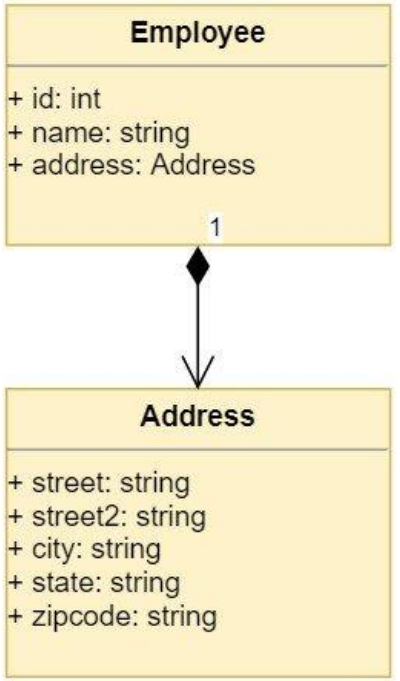




If the movie theater goes away

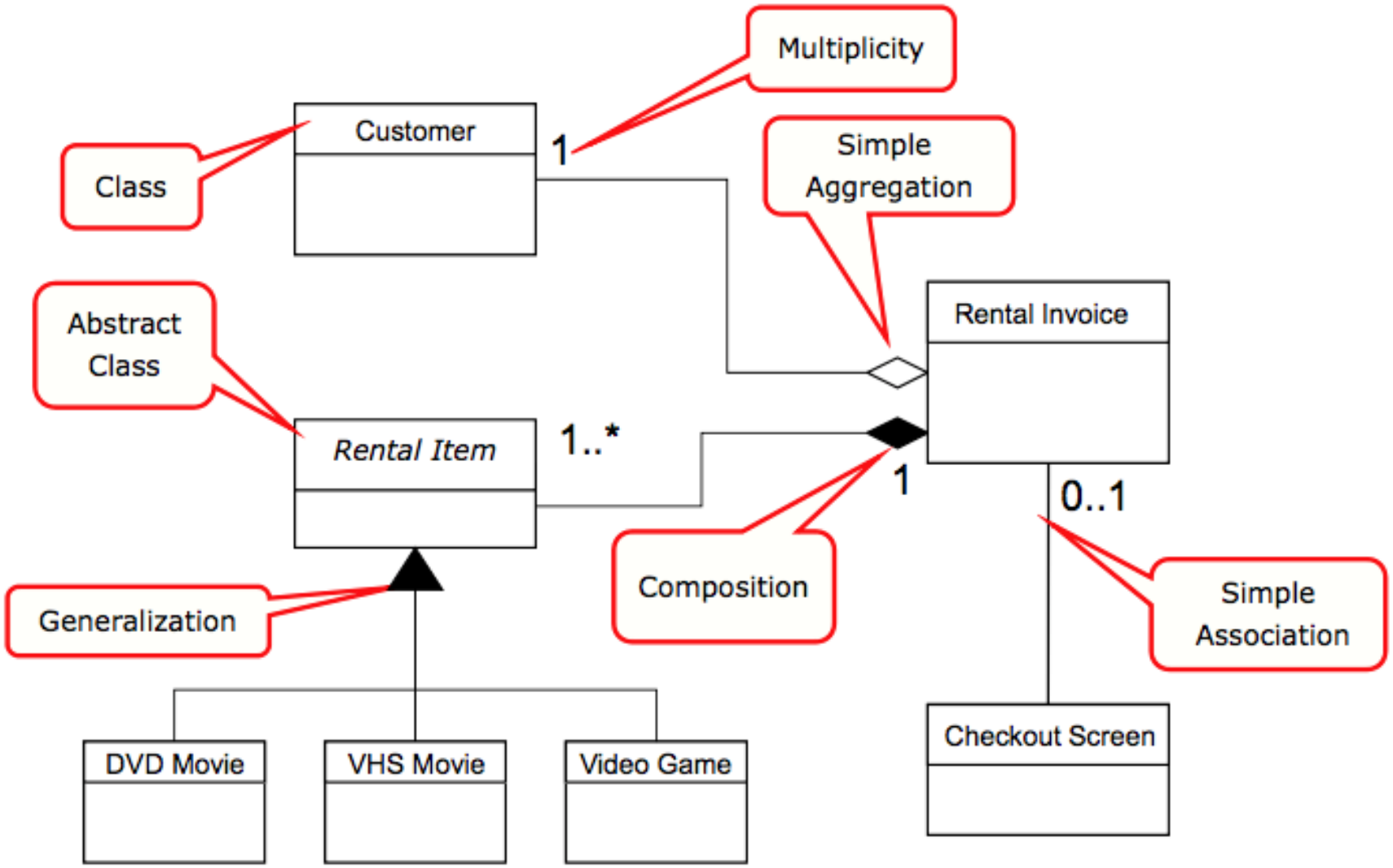
so does the box office => composition

but movies may still exist = aggregation

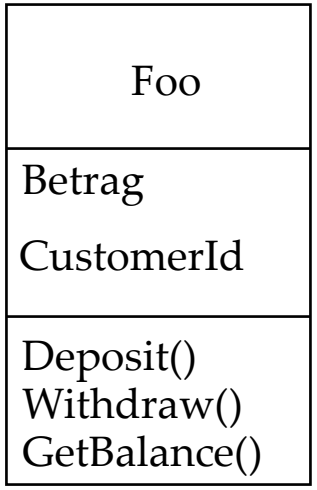


https://realpython.com/inheritance-composition-python/

Class Diagram Example : Video Store

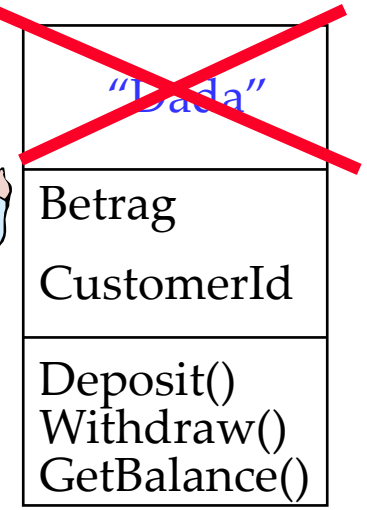


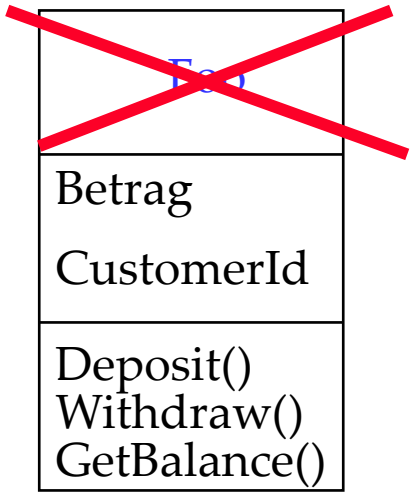
Object Modeling in Practice : Class Identification

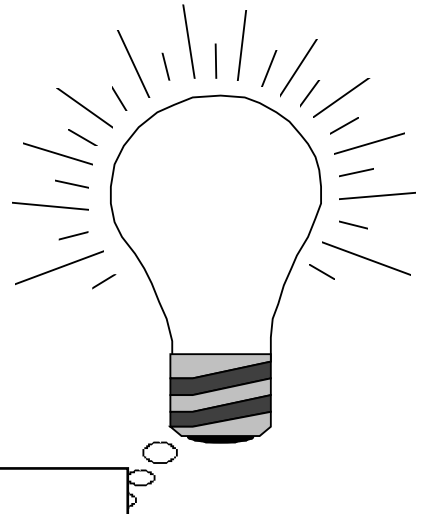


Class Identification: Name of Class, Attributes and Methods

Object Modeling in Practice : Encourage Brainstorming



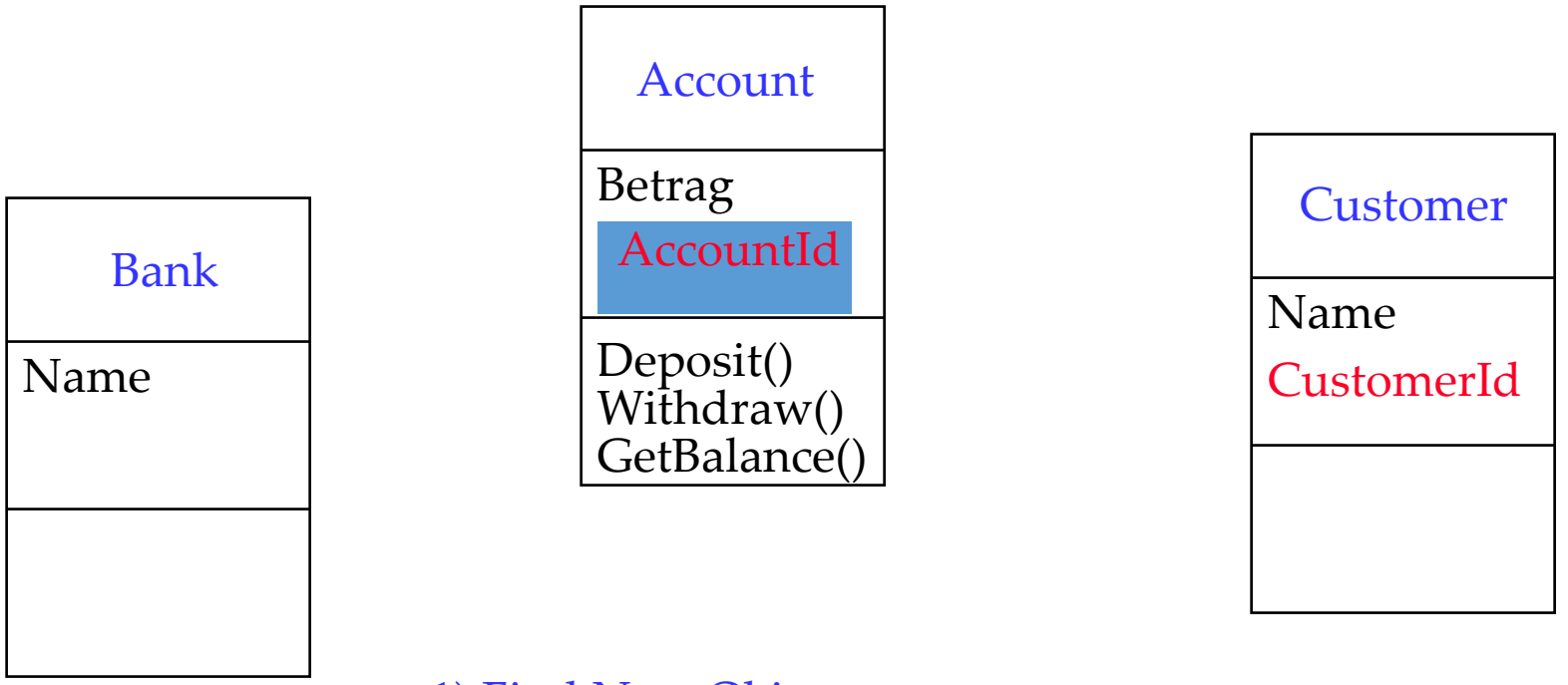




Account
Betrag
CustomerId
Deposit()
Withdraw()
GetBalance()

Naming is important!
Is Foo the right name?

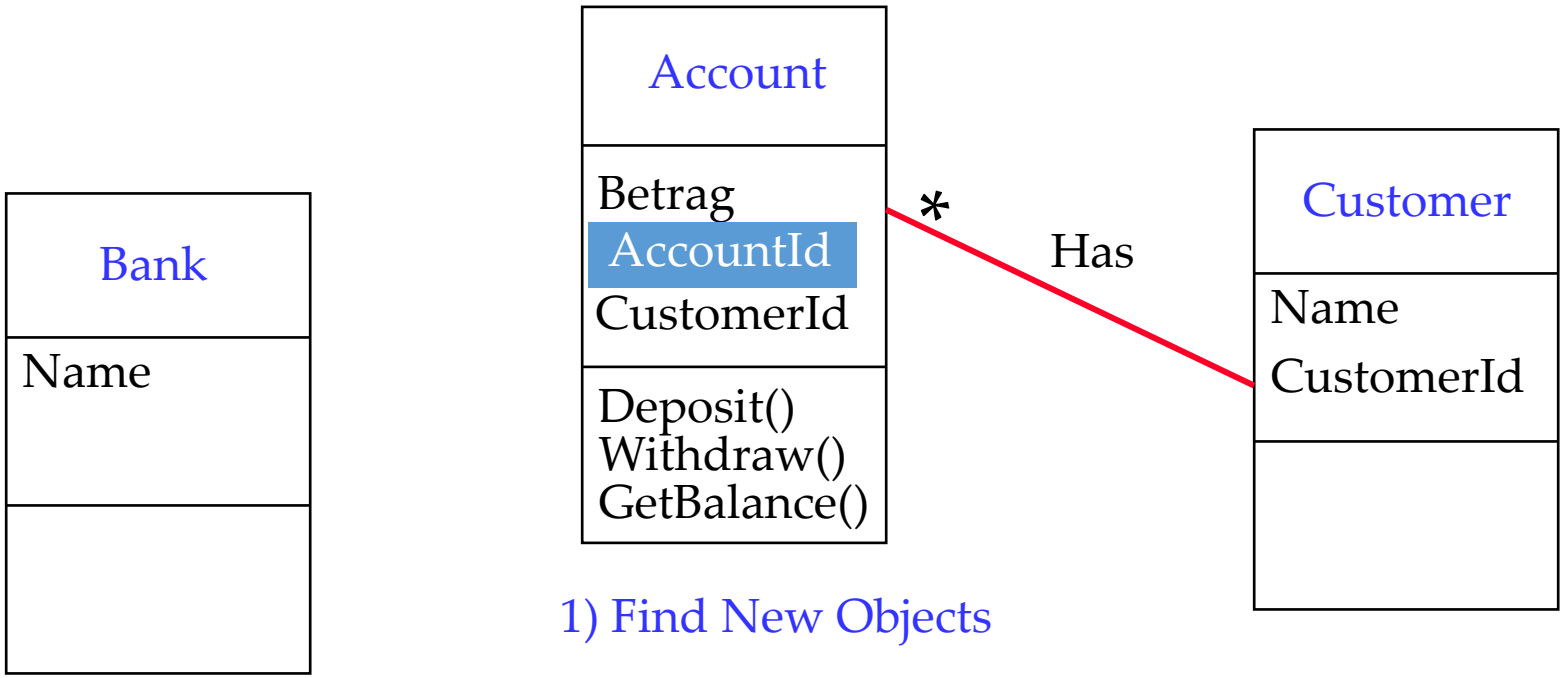
Object Modeling in Practice



1) Find New Objects

2) Iterate on Names, Attributes and Methods

Object Modeling in Practice : A Banking System



1) Find New Objects

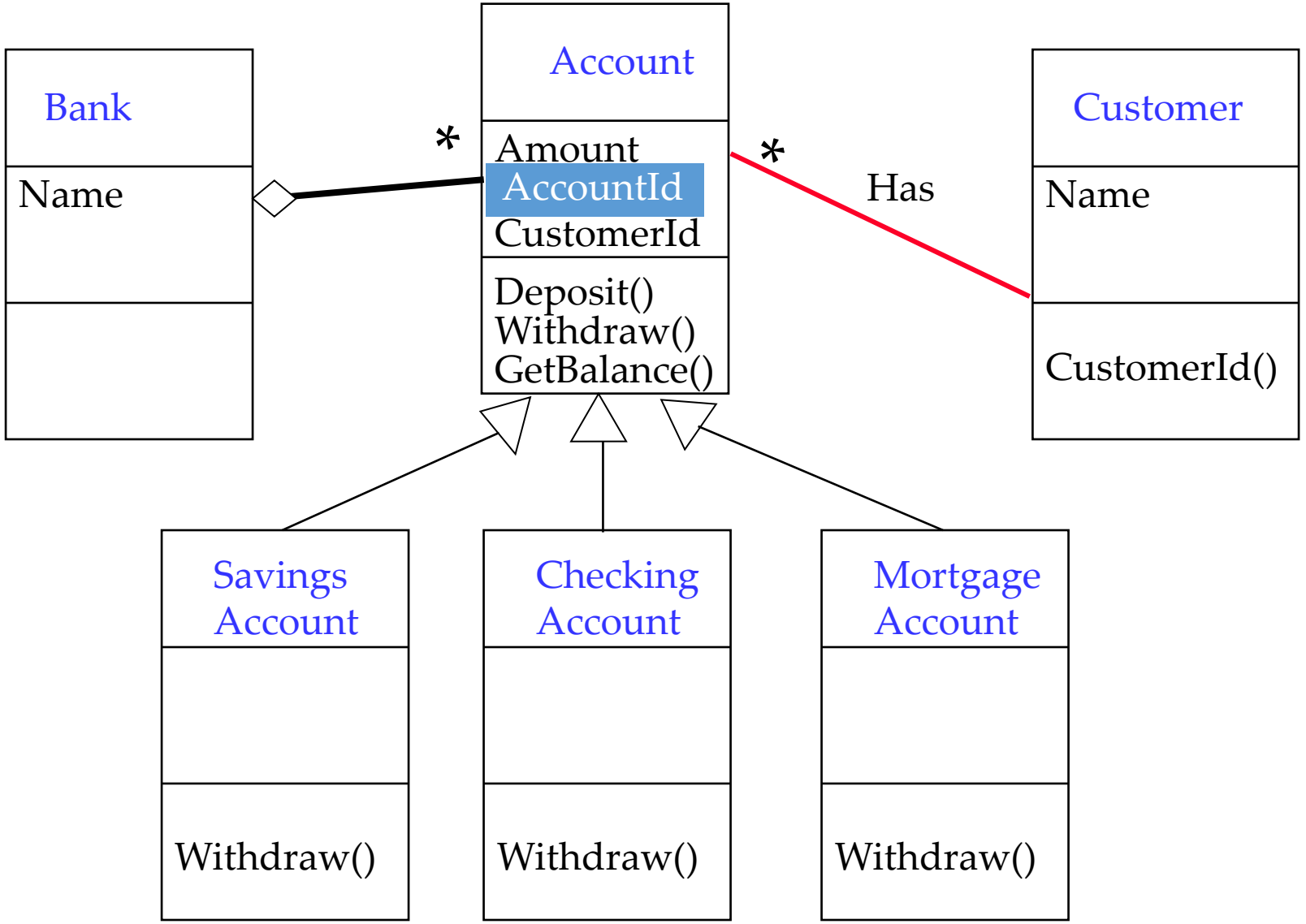
2) Iterate on Names, Attributes and Methods

3) Find Associations between Objects

4) Label the assocations

5) Determine the multiplicity of the assocations

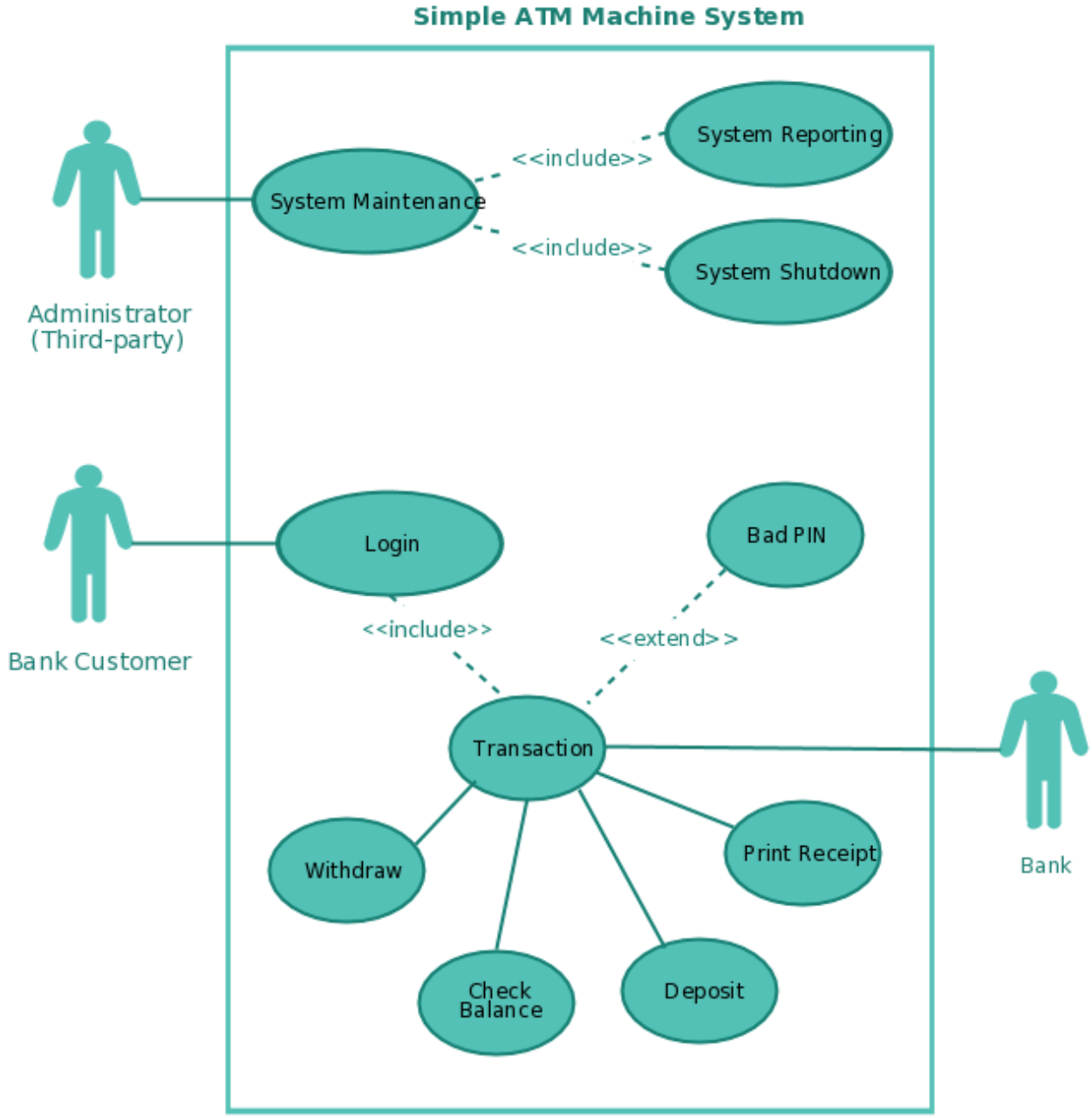
Practice Object Modeling: Iterate, Categorize!



Use Case Diagrams

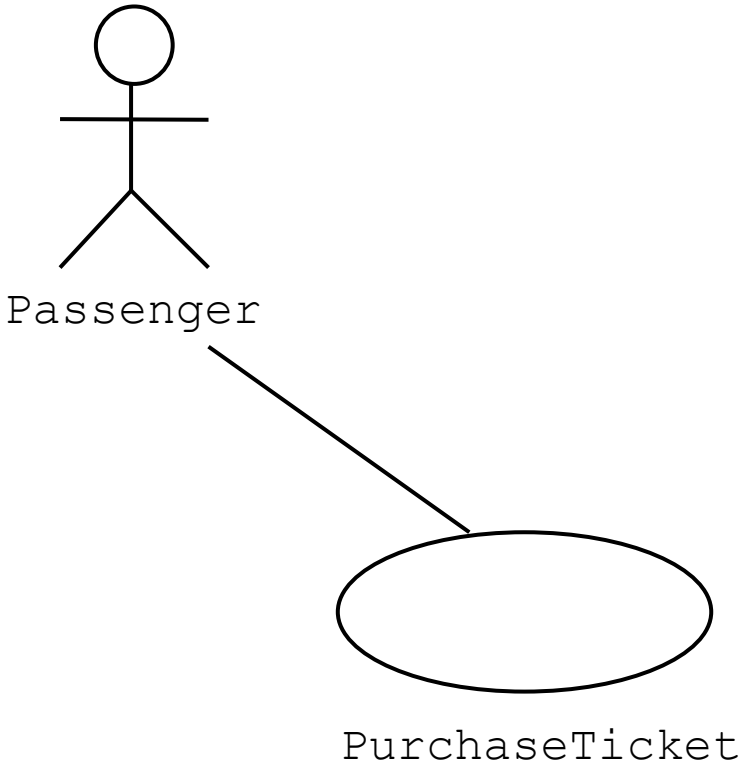
- As the most known diagram type of the behavioral UML diagrams, Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted.

- It's a great starting point for any project discussion, because you can easily identify the main actors involved and the main processes of the system. Click through to read more about use case diagram elements and/or get started instantly using our use case templates.



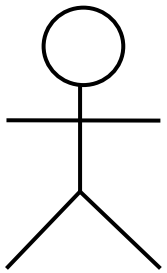
http://creately.com/blog/diagrams/uml-diagram-types-examples/

Use Case Diagrams



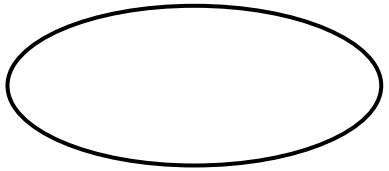
- Used during requirements elicitation to represent external behavior
- Actors represent roles, that is, a type of user of the system
- Use cases represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

Actors



- An actor models an external entity which communicates with the system:
  - User
  - External system
  - Physical environment
- An actor has a unique name and an optional description.
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates

Use Case



PurchaseTicket

- A use case represents a class of functionality provided by the system as an event flow.
- A use case consists of:
  - Unique name
  - Participating actors
  - Entry conditions
  - Flow of events
  - Exit conditions
  - Special requirements

Use Case Diagram : Example

Name: `Purchase ticket`

Participating actor: `Passenger`

Entry condition:

- `Passenger` standing in front of ticket distributor.
- `Passenger` has sufficient money to purchase ticket.

Exit condition:

- `Passenger` has ticket.

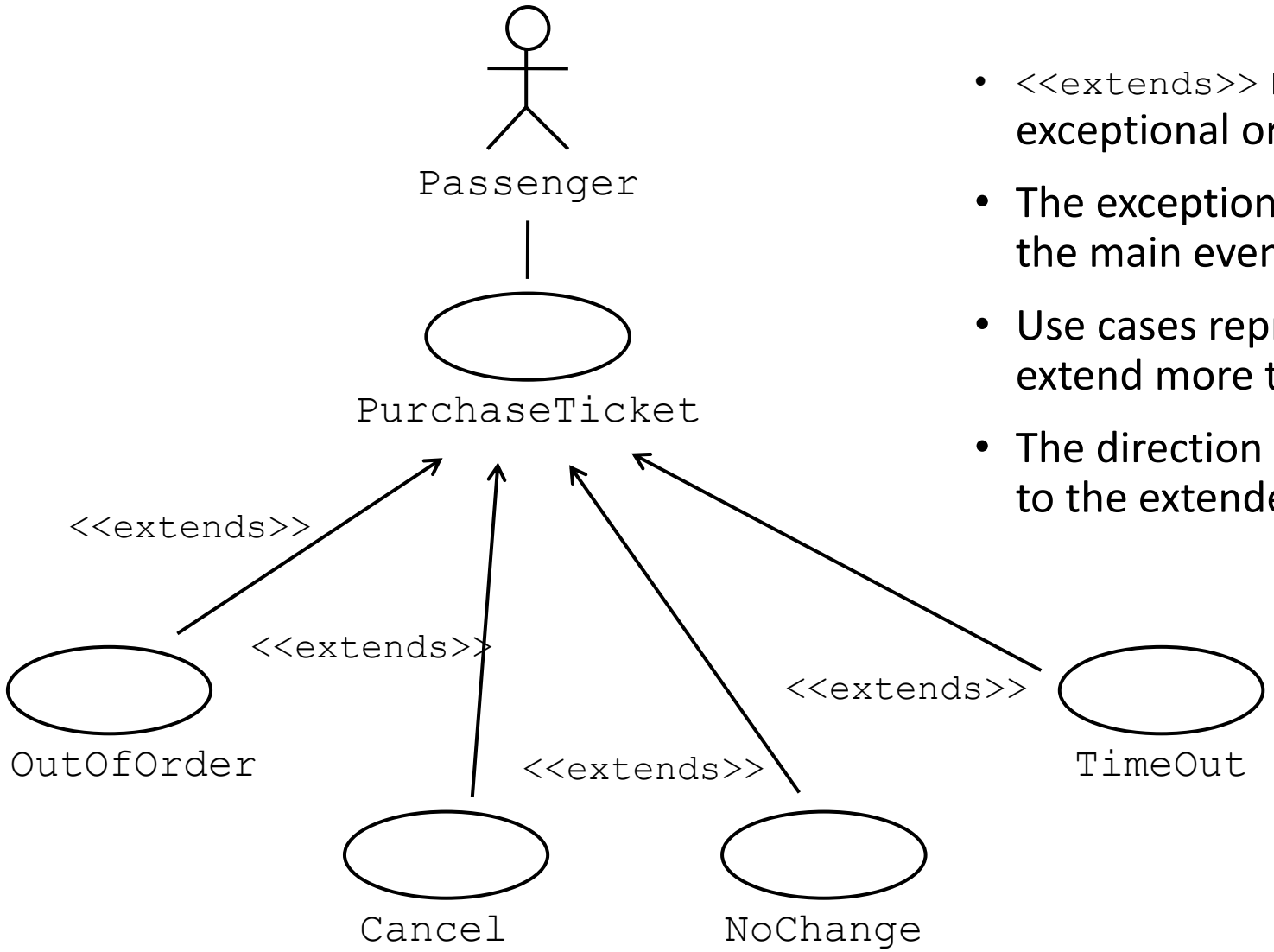
Event flow:

1. `Passenger` selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. `Passenger` inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Anything missing?

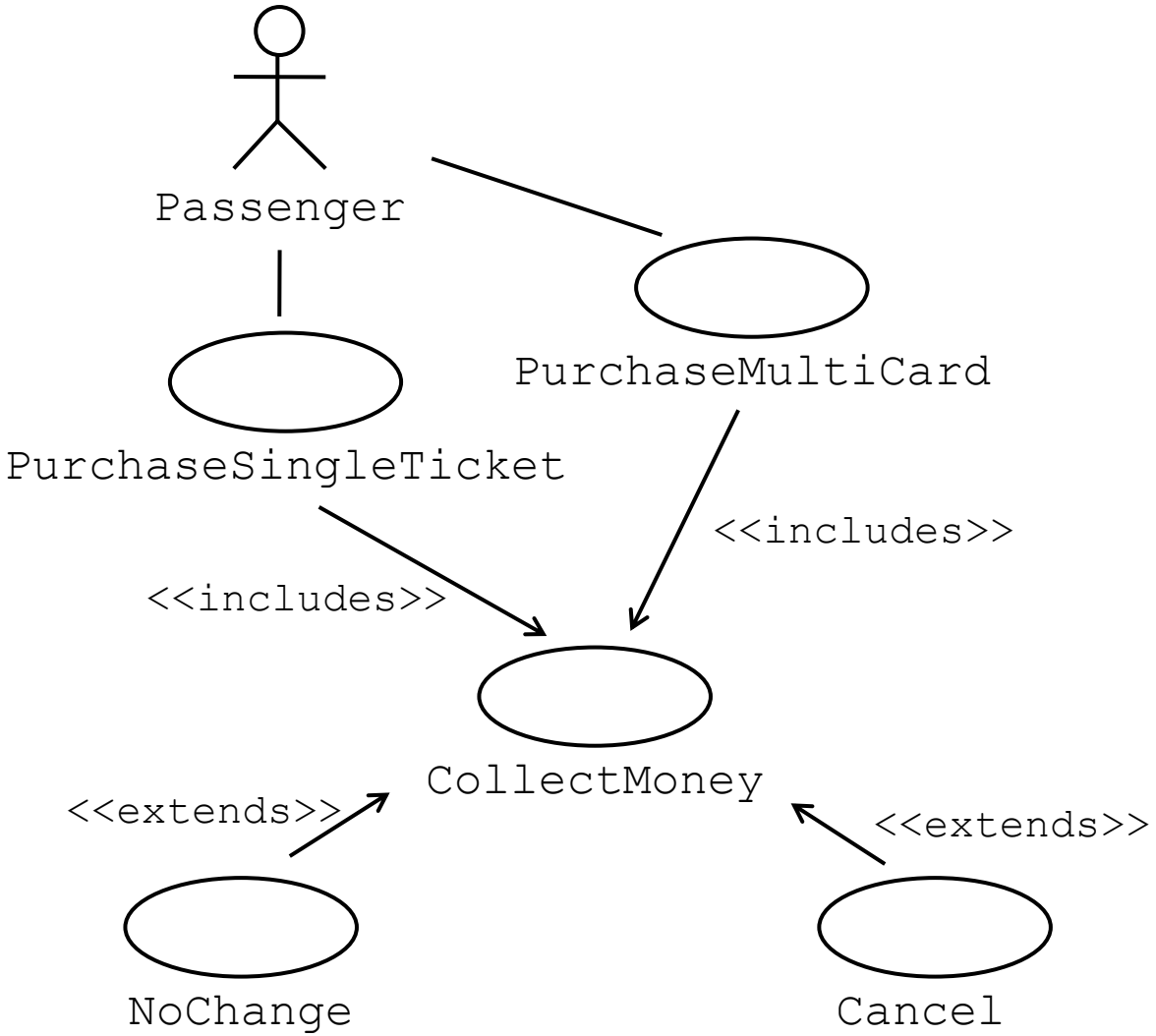
Exceptional cases!

The <<extends>> Relationship



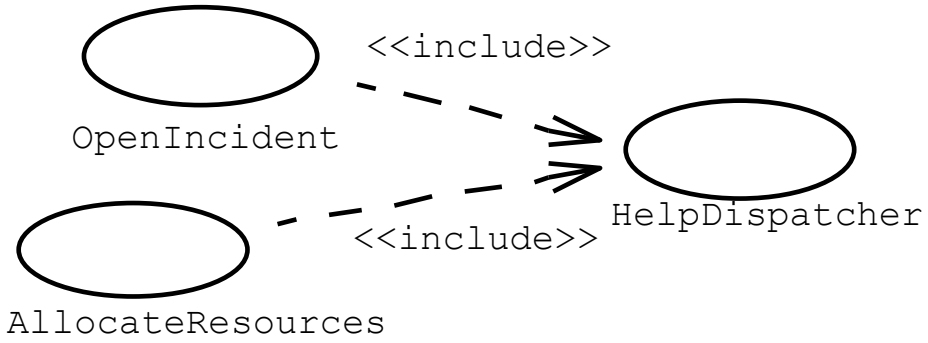
- <<extends>> relationships represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extends>> relationship is to the extended use case

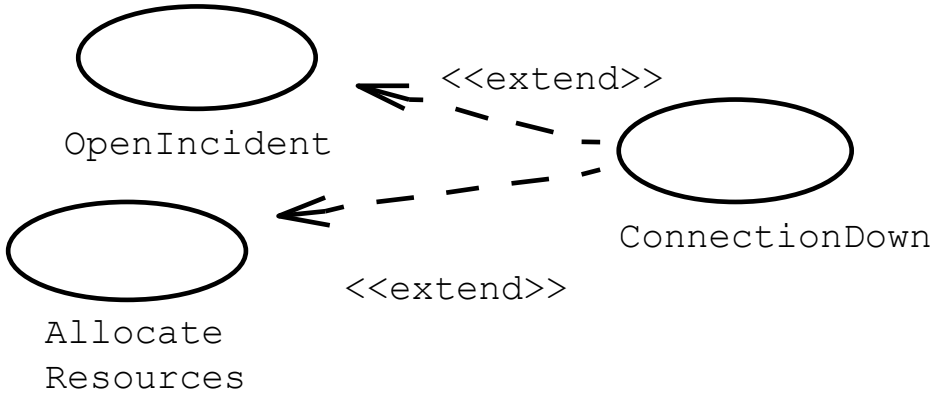
The <<includes>> Relationship

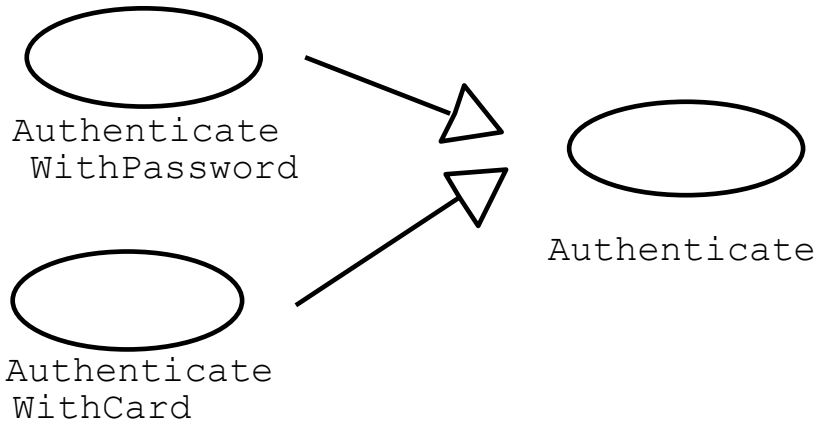


- <<includes>> relationship represents behavior that is factored out of the use case.
- <<includes>> behavior is factored out for reuse, not because it is an exception.
- The direction of a <<includes>> relationship is to the using use case (unlike <<extends>> relationships).

Use Case Diagram : Examples





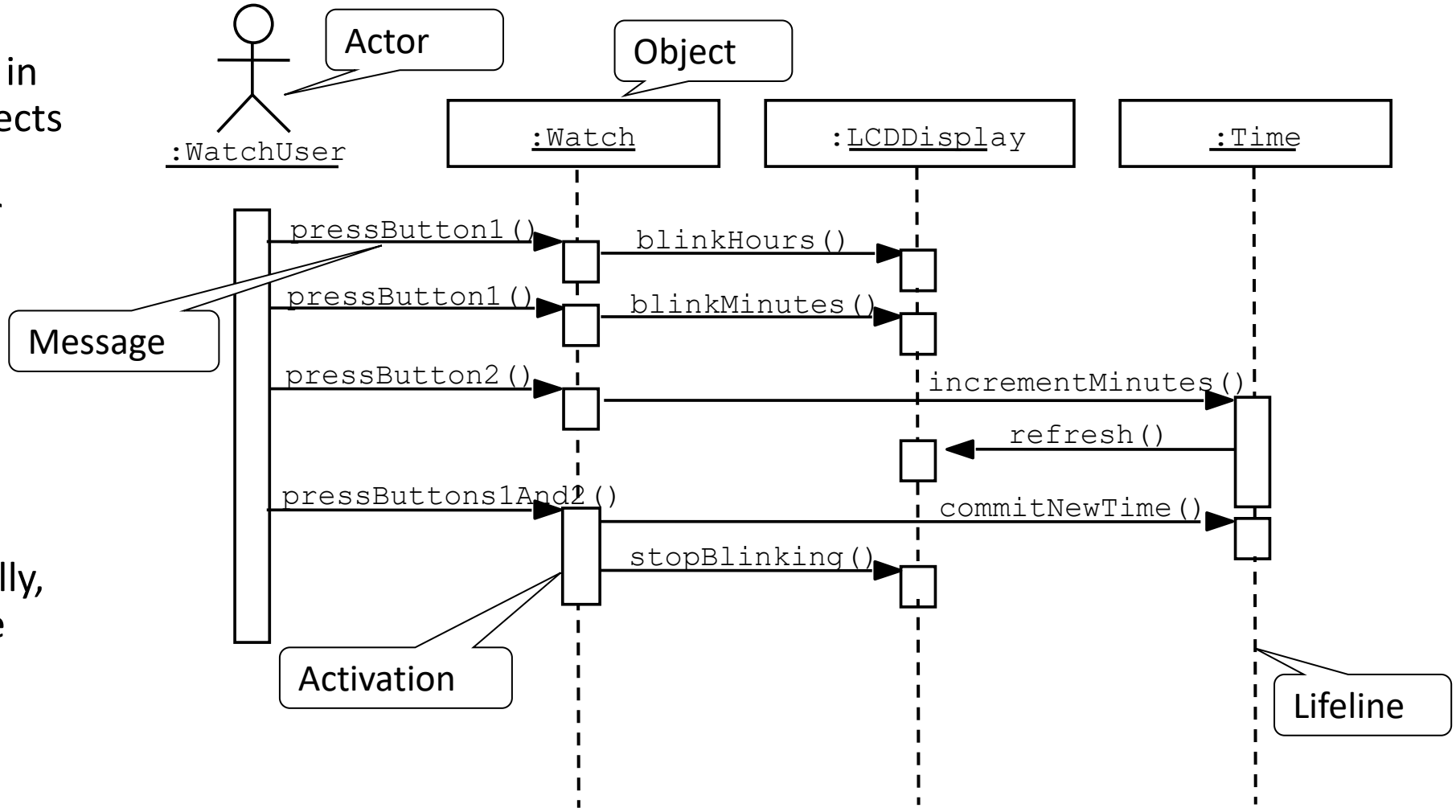


Use Case Diagrams : Summary

- Use case diagrams represent external behavior
- Use case diagrams are useful as an index into the use cases
- Use case descriptions provide meat of model, not the use case diagrams.
- All use cases need to be described for the model to be useful.

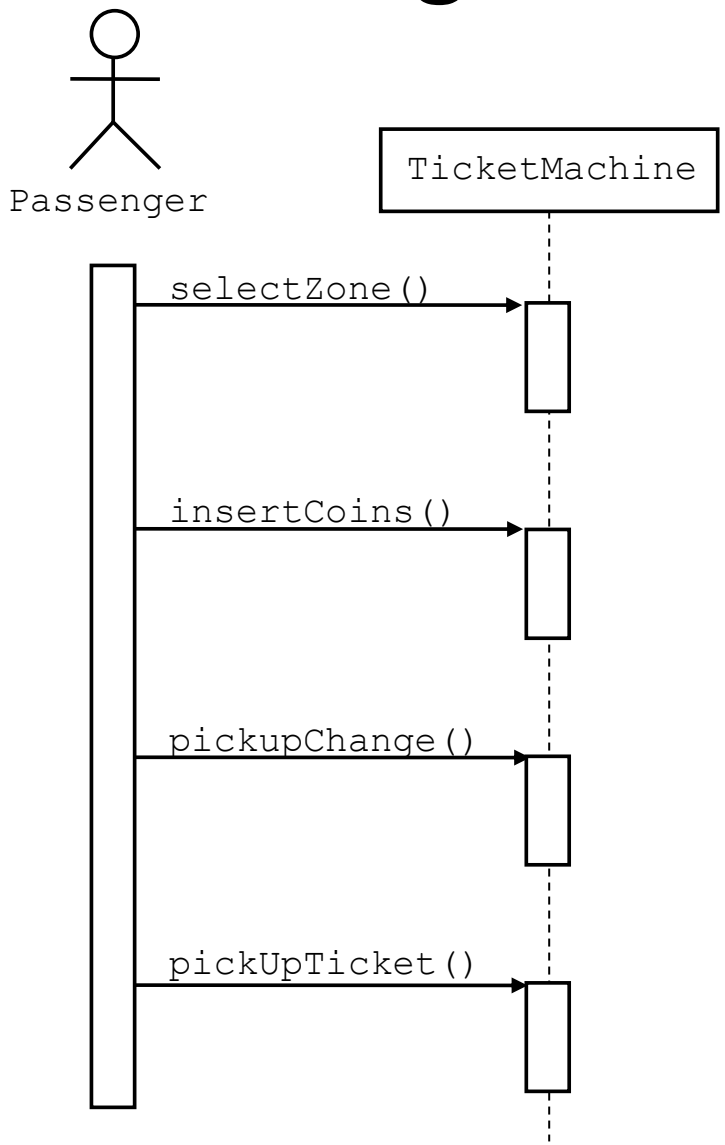
Sequence Diagram

- Sequence diagrams in UML show how objects interact with each other and the order those interactions occur.
- They show the interactions for a particular scenario.
- The processes are represented vertically, and interactions are show as arrows.



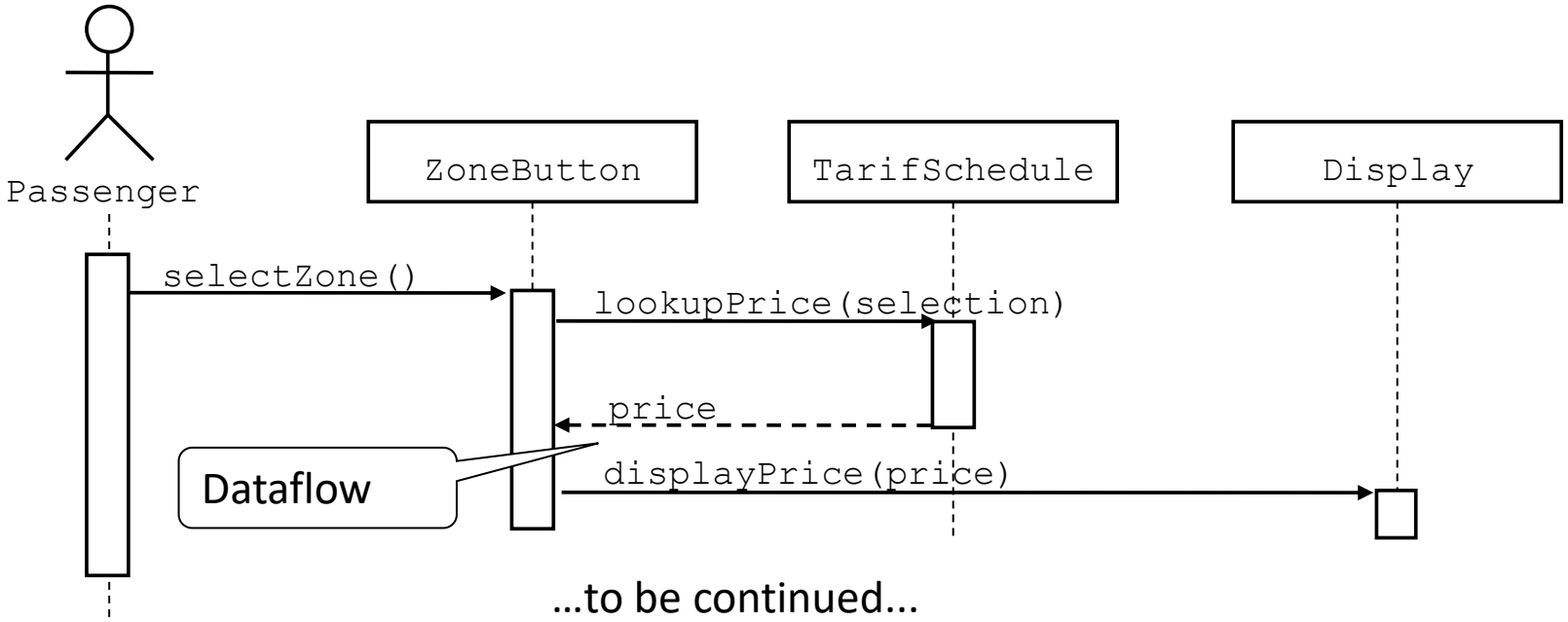
Sequence diagrams represent the behavior as interactions

Sequence Diagrams



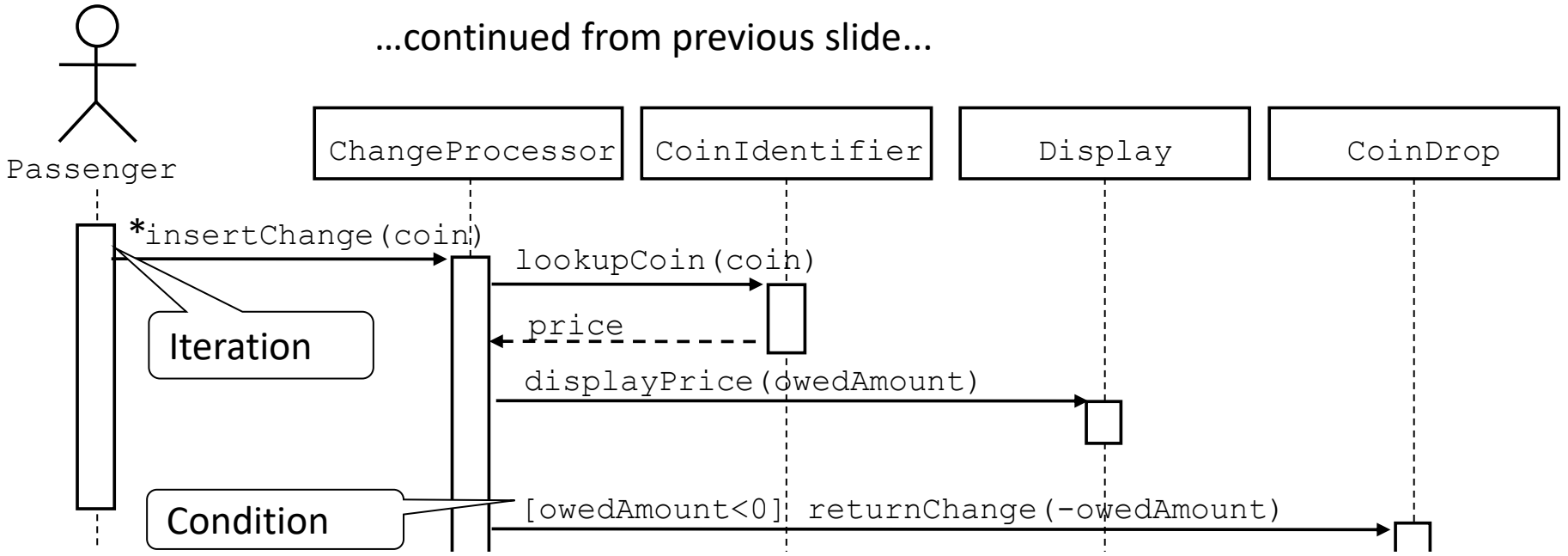
- Used during requirements analysis
  - To refine use case descriptions
  - to find additional objects (“participating objects”)
- Used during system design
  - to refine subsystem interfaces
- Classes are represented by columns
- Messages are represented by arrows
- Activations are represented by narrow rectangles
- Lifelines are represented by dashed lines

Nested Messages



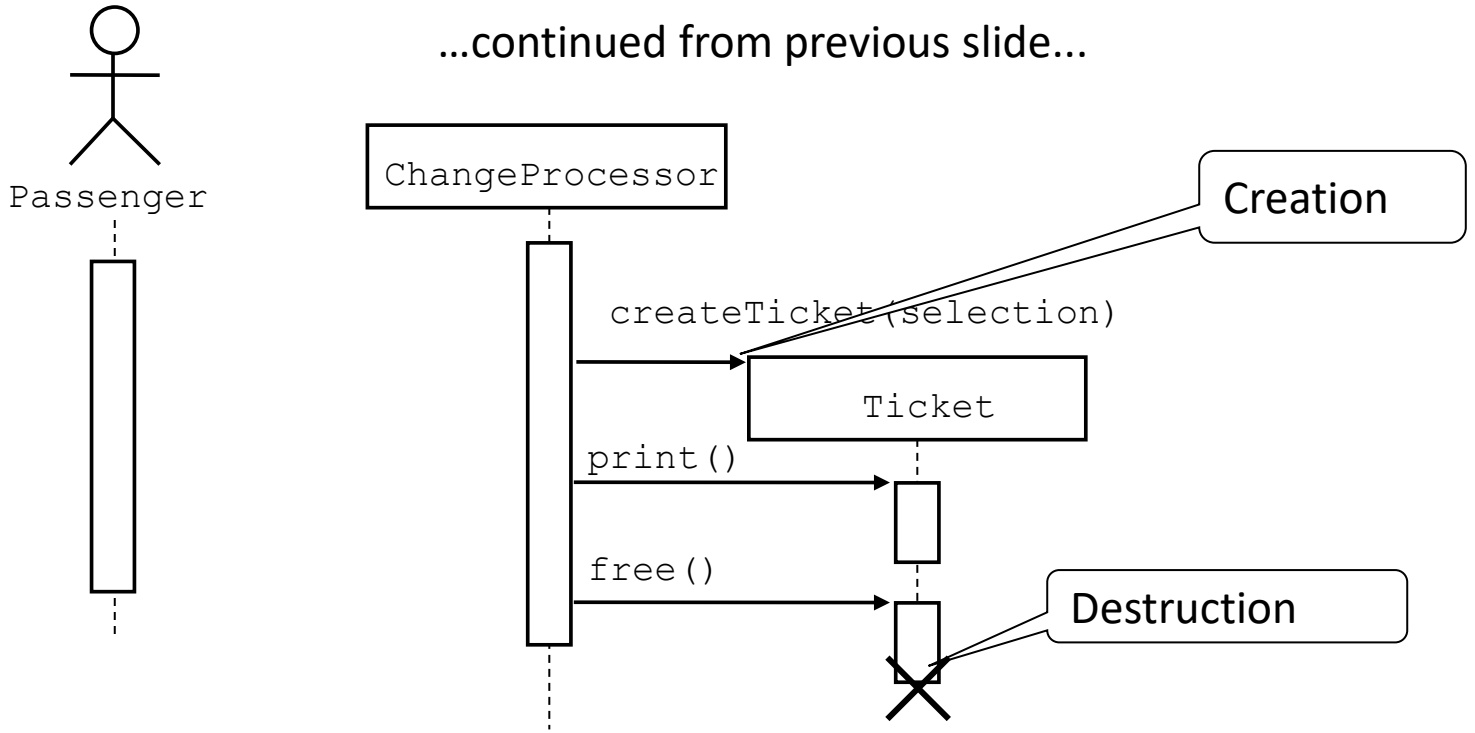
- The source of an arrow indicates the activation which sent the message
- An activation is as long as all nested activations
- Horizontal dashed arrows indicate data flow
- Vertical dashed lines indicate lifelines

Iteration & Condition



- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

Creation and Destruction



- Creation is denoted by a message arrow pointing to the object.
- Destruction is denoted by an X mark at the end of the destruction activation.
- In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

Sequence Diagram : Summary

- UML sequence diagram represent behavior in terms of interactions.
- Useful to find missing objects.
- Time consuming to build but worth the investment.
- Complement the class diagrams (which represent structure).