Modeling with UML (Use case diagrams)

Systems

- A *system* is an organized set of communicating parts
  - Natural system: A system whose ultimate purpose is may not be known
  - Engineered system: A system which is designed and built by engineers for a specific purpose
- The parts of the system can be considered as systems again
  - In this case we call them *subsystems*

Examples of natural systems: Universe, earth, ocean

Examples of engineered systems: Airplane, watch, GPS

Examples of subsystems: Jet engine, battery, satellite.

Systems, Models and Views

A ***model*** is an abstraction describing a system or a subsystem

A ***view*** depicts selected aspects of a model

A ***notation*** is a set of graphical or textual rules for depicting models and views

System: Airplane

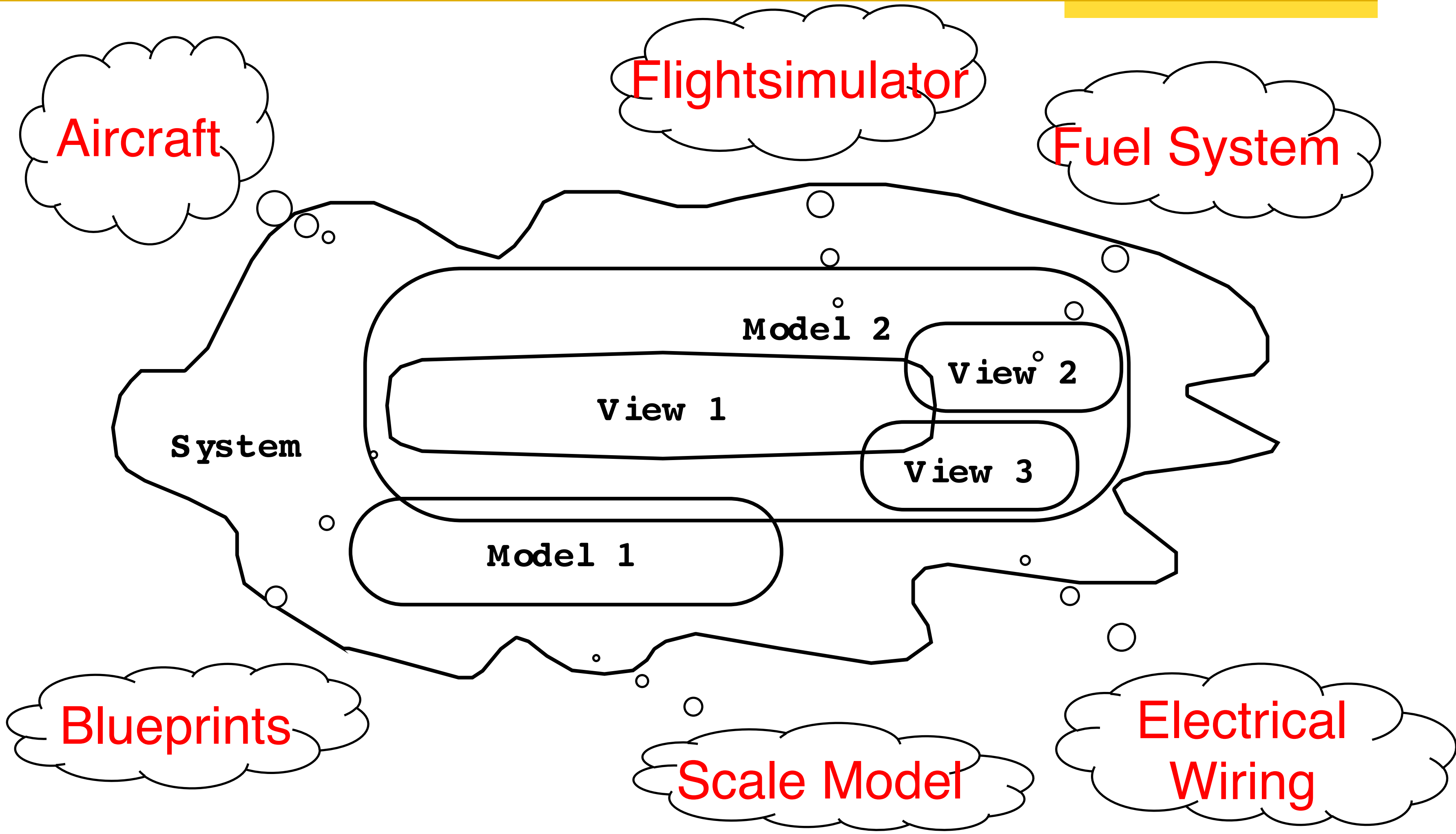
- **Models:** Flight simulator, Scale model
- **Views:**

  Blueprint of the airplane components

  Electrical wiring diagram, Fuel system

Systems, Models and Views

("Napkin" Notation)



Views and models of a complex system usually overlap

What is UML? Unified Modeling Language

- Convergence of different notations used in object-oriented methods, mainly - OMT (James Rumbaugh and collegues), OOSE (Ivar Jacobson), Booch (Grady Booch)
- They also developed the Rational Unified Process, which became the Unified Process in 1999

25 year at GE Research, where he developed OMT, joined (IBM) Rational in 1994, CASE tool OMTool

At Ericsson until 1994, developed use cases and the CASE tool Objectory, at IBM Rational since 1995, http://www.ivarjacobson.com

Developed the Booch method ("clouds"), ACM Fellow 1995, and IBM Fellow 2003 http://www.booch.com/

What is UML?

- UML (Unified Modeling Language)
  - Nonproprietary standard for modeling software systems, OMG
  - Convergence of notations used in object-oriented methods
    - OMT (James Rumbaugh and collegues)
    - Booch (Grady Booch)
    - OOSE (Ivar Jacobson)
- Current Version: UML 2.5
  - Information at the OMG portal http://www.uml.org/
- Commercial tools: Rational (IBM), Together (Borland), Visual Architect (busines processes, BCD)
- Open Source tools: ArgoUML, StarUML, Umbrello
- Commercial and Opensource: PoseidonUML (Gentleware)

UML: First Pass

- You can model 80% of most problems by using about 20% UML
- We’ll learn those 20%

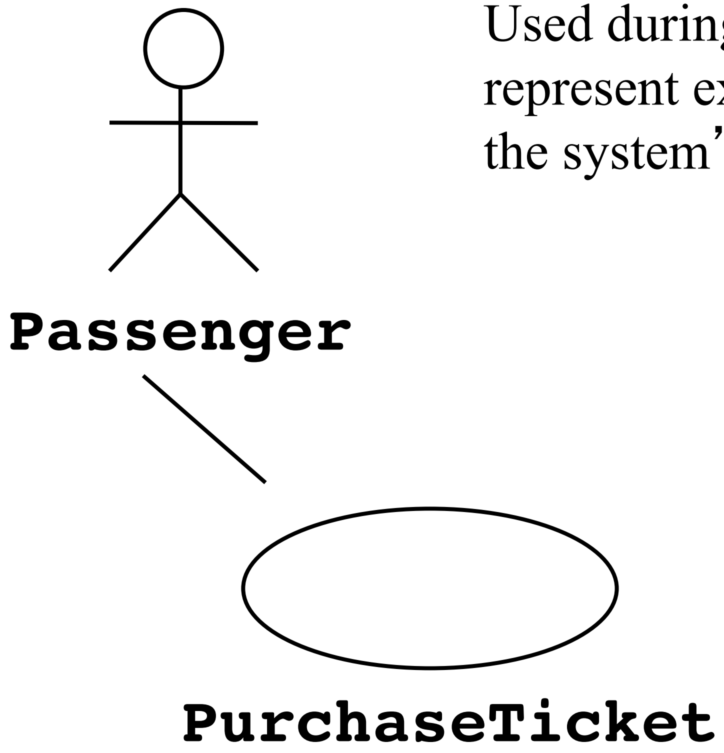
UML First Pass

- Use case diagrams: Describe the functional behavior of the system as seen by the user
- Class diagrams: Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams: Describe the dynamic behavior between objects of the system
- State chart diagrams: Describe the dynamic behavior of an individual object
- Activity diagrams: Describe the dynamic behavior of a system, in particular the workflow.

UML Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities and drawn as rectangles or ovals
  - $\boxed{\text{Rectangles}}$ denote classes or instances
  - Ovals denote functions
- Names of Classes are not underlined
  - SimpleWatch
  - Firefighter
- Names of Instances are underlined
  - <u>myWatch:SimpleWatch</u>
  - <u>Joe:Firefighter</u>
- An edge between two nodes denotes a relationship between the corresponding entities

UML Use Case Diagrams



Used during requirements elicitation and analysis to represent external behavior ("visible from the outside of the system")

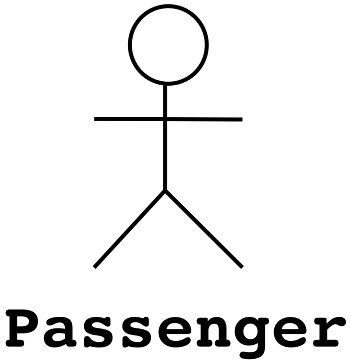
An ***Actor*** represents a role, that is, a type of user of the system

A ***use case*** represents a class of functionality provided by the system

Use case model:

The set of all use cases that completely describe the functionality of the system.

Actors



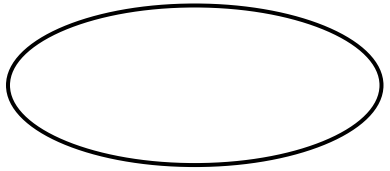
- An actor is a model for an external entity which interacts (communicates) with the system:
  - User
  - External system (Another system)
  - Physical environment (e.g. Weather)
- An actor has a unique name and an optional description
- Examples:
  - **Passenger**: A person in the train
  - **GPS satellite**: An external system that provides the system with GPS coordinates.

Optional Description

Name

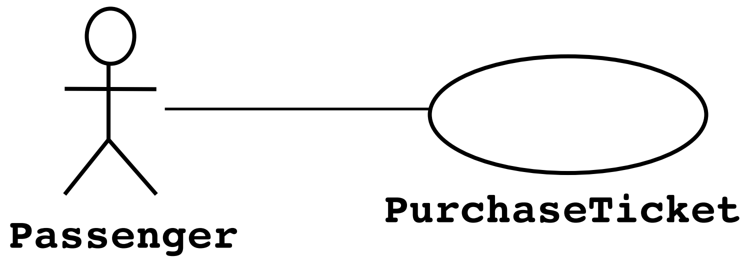
Use Case

- A use case represents a class of functionality provided by the system
- Use cases can be described textually, with a focus on the event flow between actor and system
- The textual use case description consists of 6 parts:
  1. Unique name
  2. Participating actors
  3. Entry conditions
  4. Exit conditions
  5. Flow of events
  6. Special requirements.



PurchaseTicket

Textual Use Case Description Example



1. Name: Purchase ticket

2. Participating actor: Passenger

3. Entry condition:

- *Passenger* stands in front of ticket distributor
- *Passenger* has sufficient money to purchase ticket

4. Exit condition:

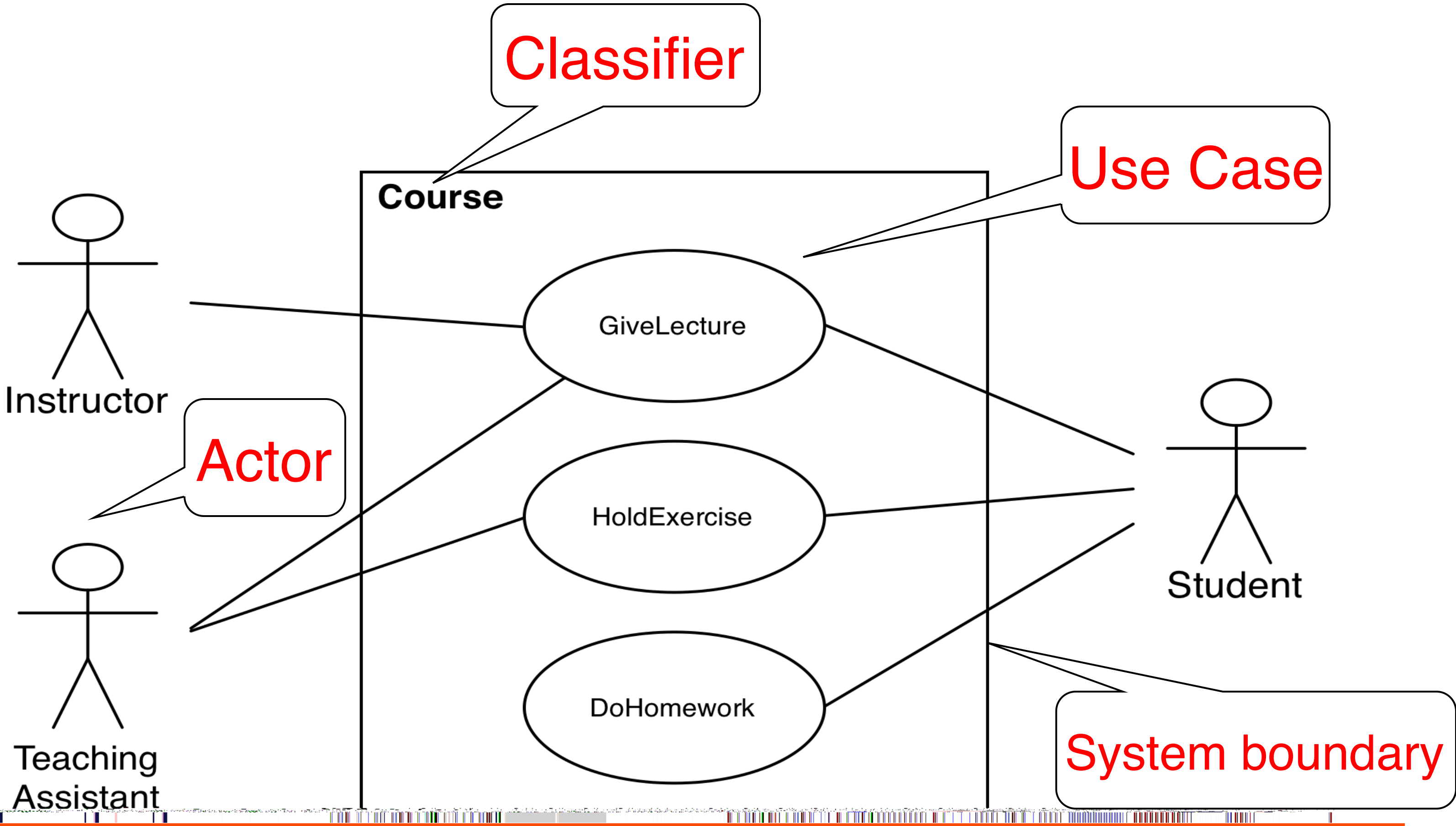
- *Passenger* has ticket

5. Flow of events:

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

6. Special requirements: None.

UML first pass: Use case diagrams

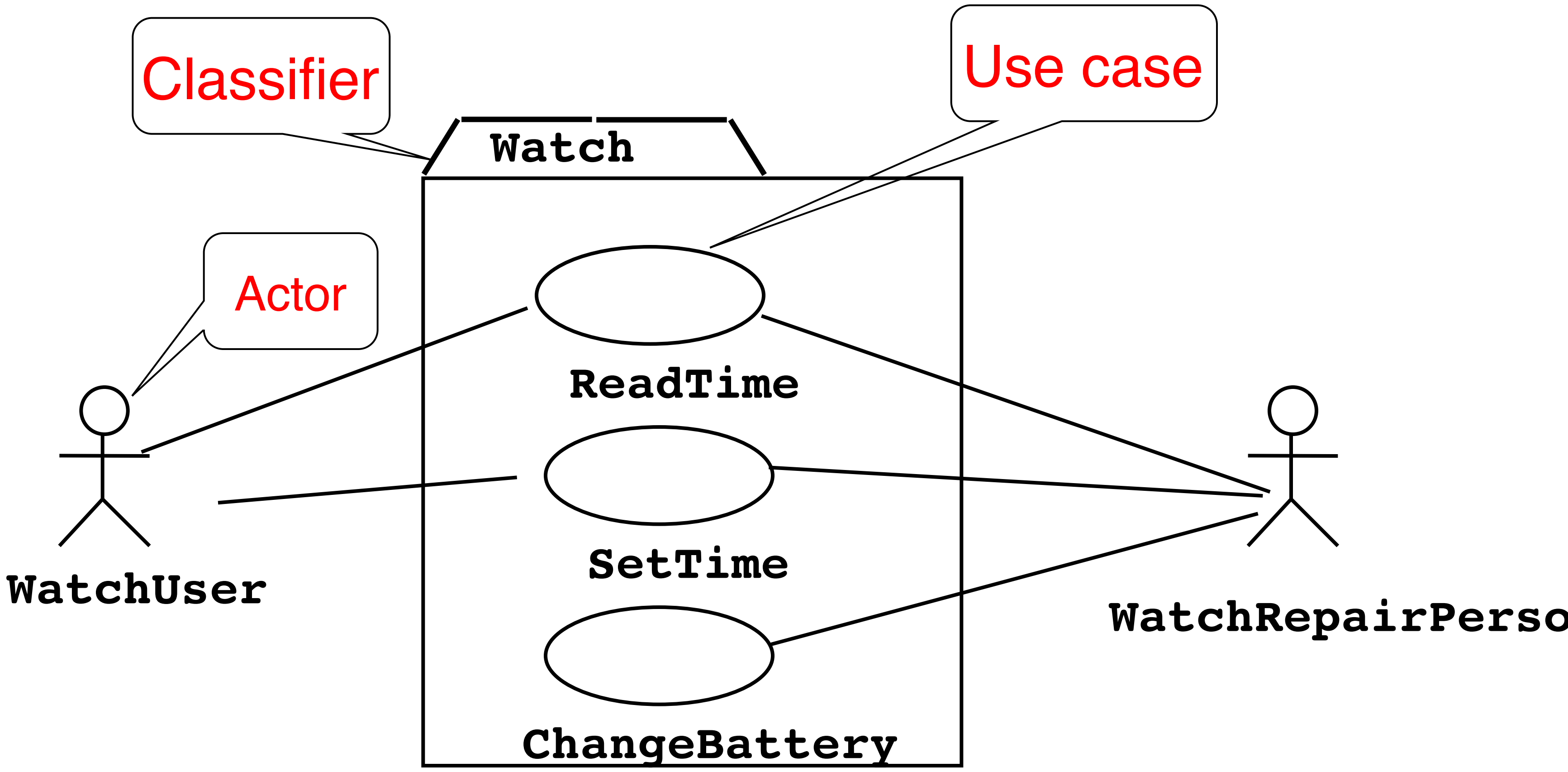


Use case diagrams represent the functionality of the system from user’s point of view

Exercise: Use Case Diagram

- Draw a Use Case diagram for a SimpleWatch. A user of the watch can read time and set time. The battery of the watch can be changed by a repair person.

Exercise: Use Case Diagram

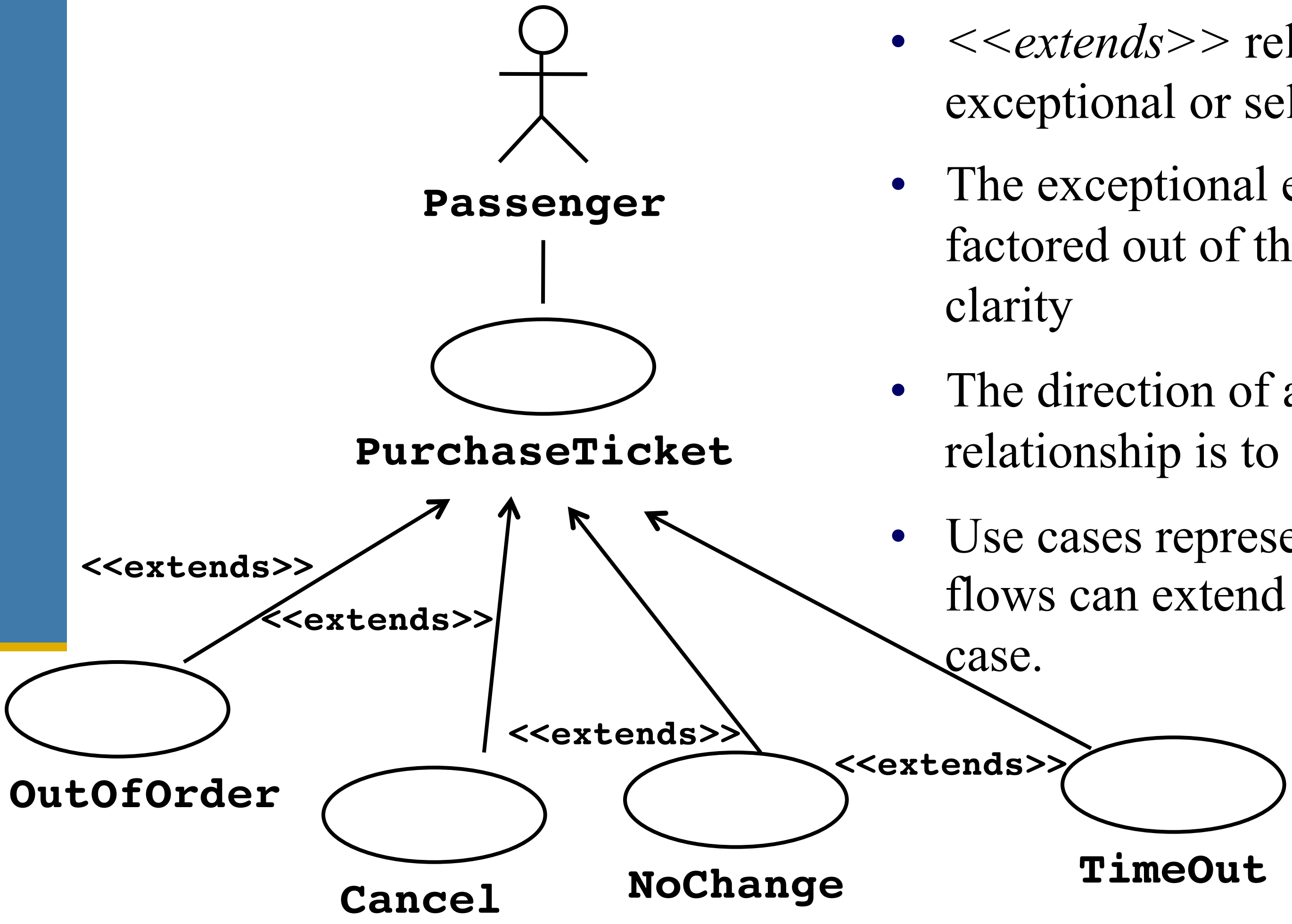


Use case diagrams represent the functionality of the system from user’s point of view

Uses Cases can be related

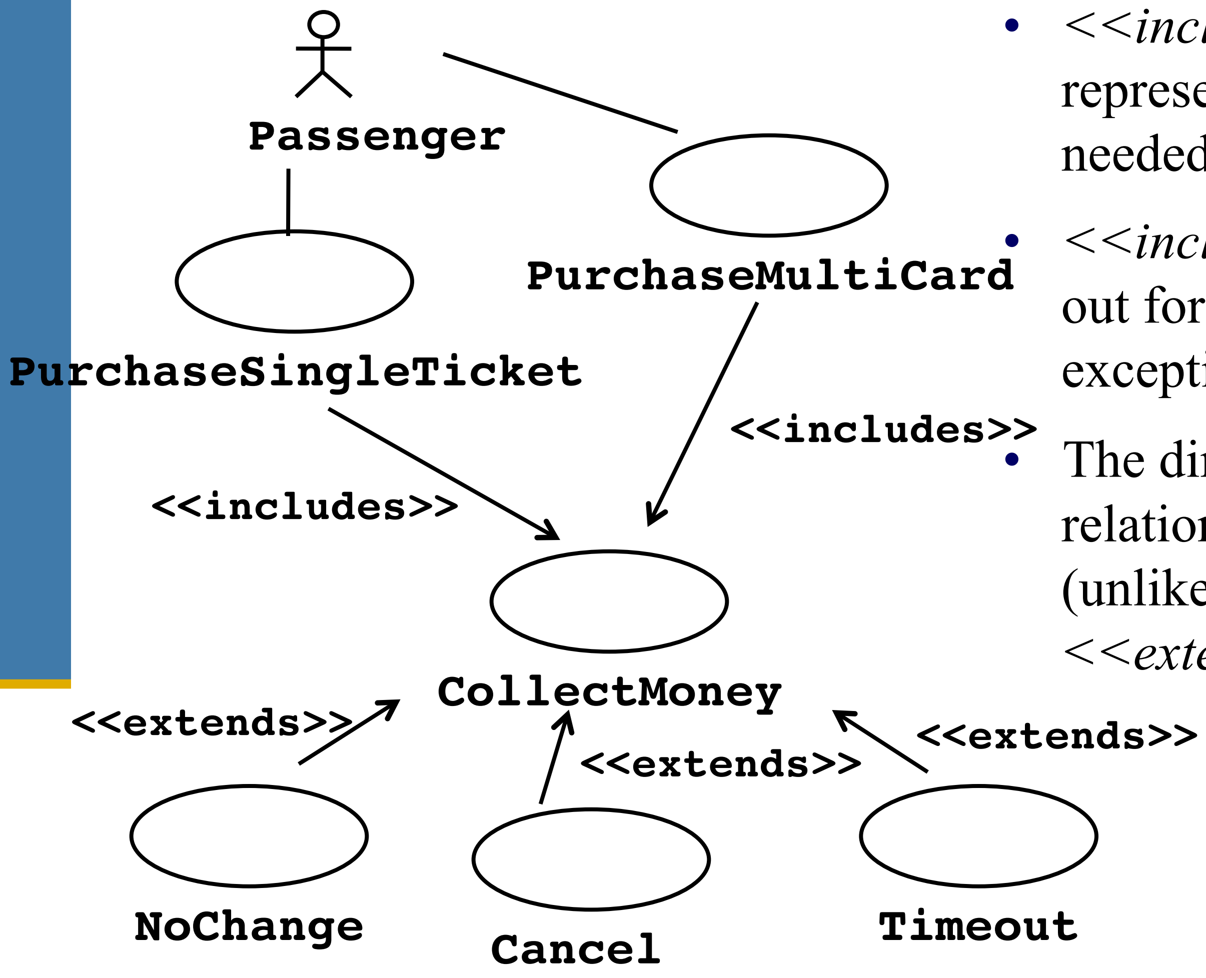
- Extends Relationship
  - To represent seldom invoked use cases or exceptional functionality
- Includes Relationship
  - To represent functional behavior common to more than one use case.

The <<*extends*>> Relationship



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

The <<*includes*>> Relationship



- <<*includes*>> relationship represents common functionality needed in more than one 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 the direction of the <<*extends*>> relationship).