CMPT 276 Class 12: System Modelling

Dr. Jack Thomas
Simon Fraser University
Fall 2020

Today's Topics

1. Why **model a system**?
2. How can we model...
   A. the **context** of a system?
   B. the **interactions** with the system?
   C. the **structure** of a system?
   D. the **behaviour** of a system?
3. Can we use models to **generate** a system?

System Modelling

- The process of developing **abstract models of a system**, where each model shows **a different perspective of the same system**.
- Usually models are graphical, such as the **Unified Modelling Language** (UML).
- Modelling leaves out details, the challenge is including only the right details.

A Word of Warning on Models

- “The map is not the territory,”
  - Alfred Korzybski

- Like plans, the value and purpose of models are hotly debated.

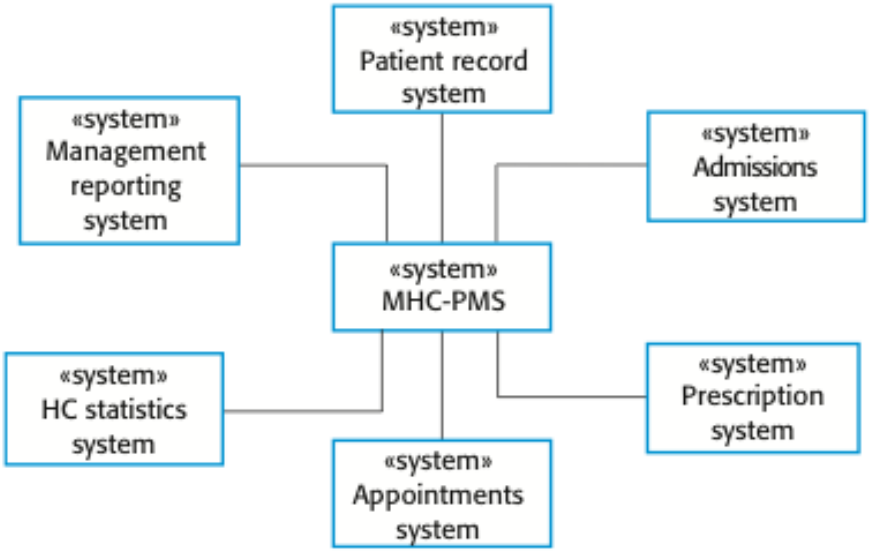
- Data-driven and machine learning approaches claim to be moving us past models.

System Perspectives

- There are many perspectives on the same system. For example a couch has concept art, a design sketch, blueprints, assembly diagrams, and more.
- **External Perspective**: Model the environment (context) where system is used.
- **Interaction Perspective**: Model the interactions between a system and its environment.
- **Structural Perspective**: Model the organization of a system or structure of its data.
- **Behavioural Perspective**: Model the dynamic behaviour of the system and how it responds to events

Context Models

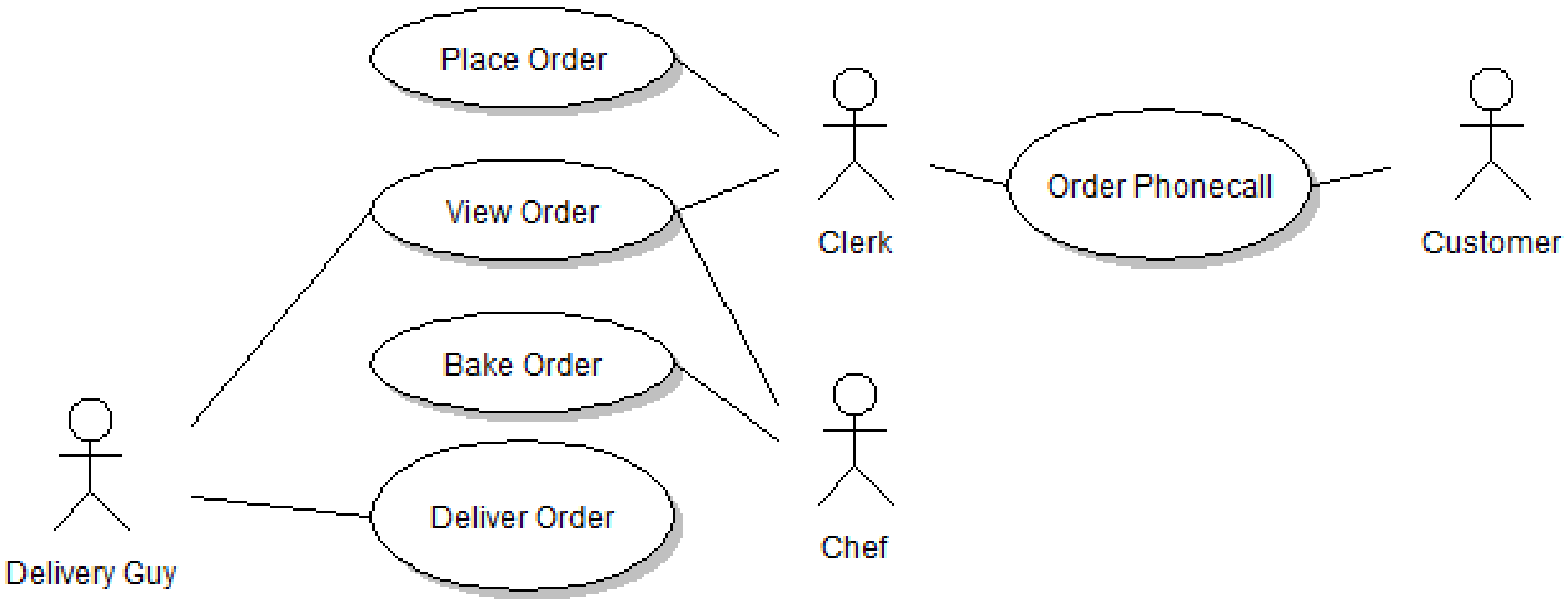
- Models what lies outside the system boundaries.
  - Show other systems which use or are used by the new system.
  - Does not show the nature of the relationships: "who uses whom?"
- Position of the system boundary has a profound effect on system requirements, but is a ‘political’ judgment



Use Case Modelling

- Each use case represents a task with an **external interaction** of value to the actor.
- Use cases show a very high-level view
  - **Actors** (stick-figures): people or other systems.
  - **Actions** (ellipses): the interaction.
- Can complete the model with a text description of the interaction.
- Does **not** show the sequence of actions.

Use Case Diagram for Ordering Pizza



Note: The system being developed isn't shown on the diagram, it IS the diagram.
Note 2: Tip your pizza guy, the world is on fire.

Use Case Exercise: CourSys

- Draw a UML Use Case diagram of CourSys for the following:
  - **Actions**: Grade submission, Submit, Configure class, View grade
  - **Users**: Student, Instructor, TA, Admin

Structural Models

- Structural models of software show the organization of a system in terms of **its components and their relationships**.
- **Static** structural models shows the structure of the system design.
  - Ex: Classes
- Use structural models of a system when discussing and designing the system architecture.

UML Class Diagram

- A diagram showing classes and relationships between them.

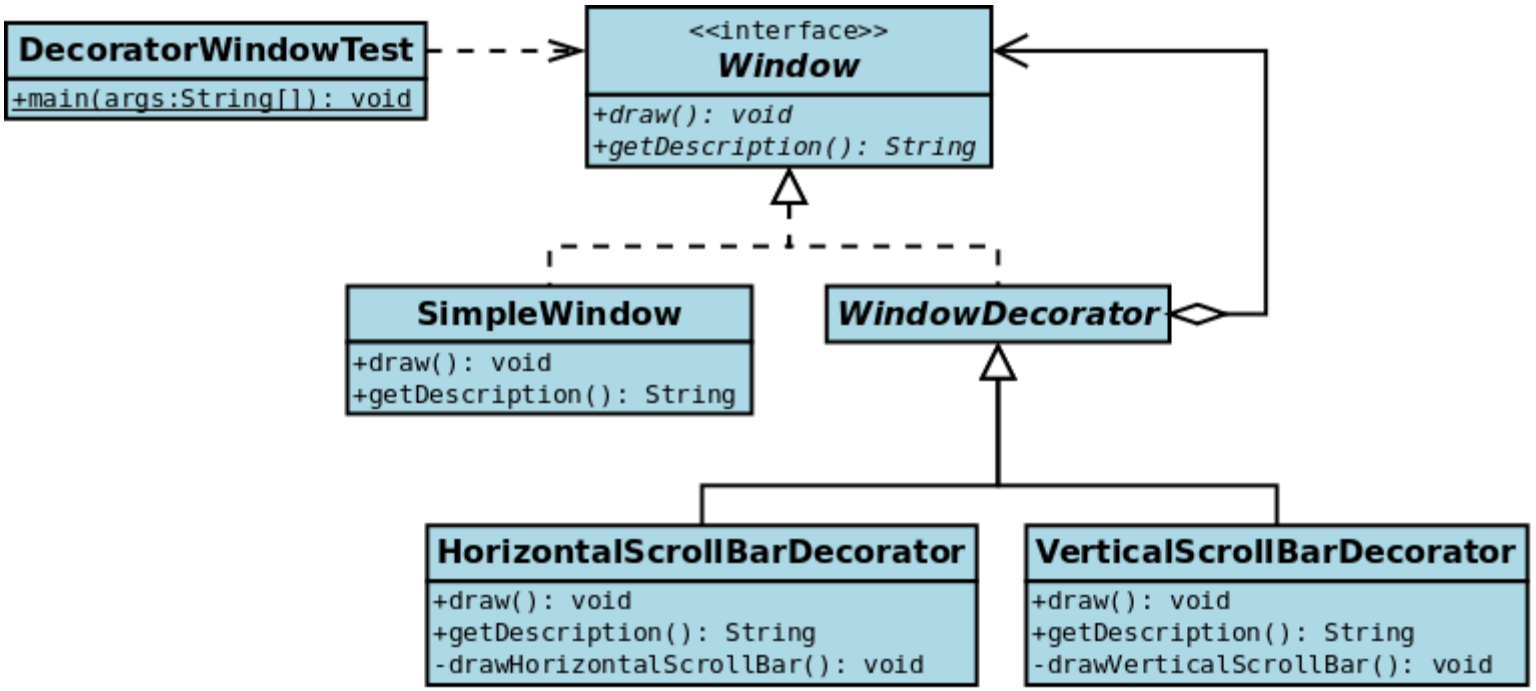
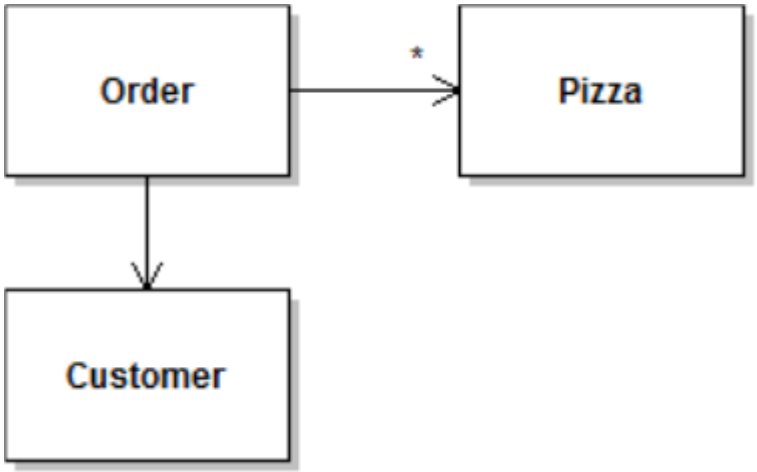


Image Credit: https://en.wikipedia.org/wiki/Class_diagram

Relationship: Aggregation

- The “Has-A” Relationship: Shows an object **composed of** other objects. Ex: a cell-phone has a screen, or has many buttons.

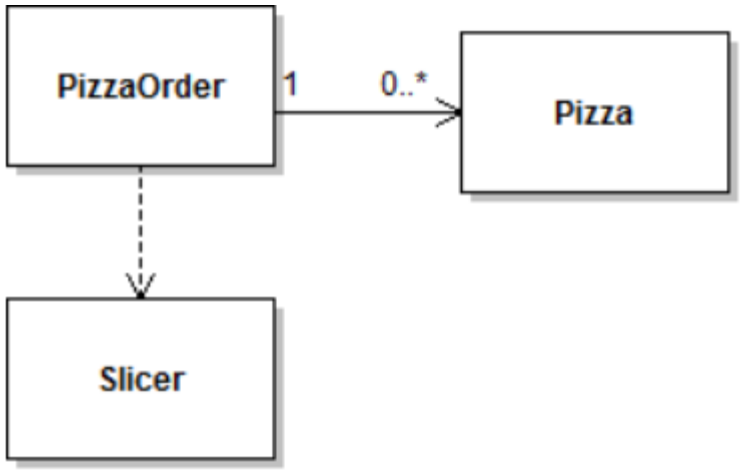


- Show the number, like 1, or 0..1, or *.
- Hint: This is usually for an object’s fields.

Relationship: Dependency

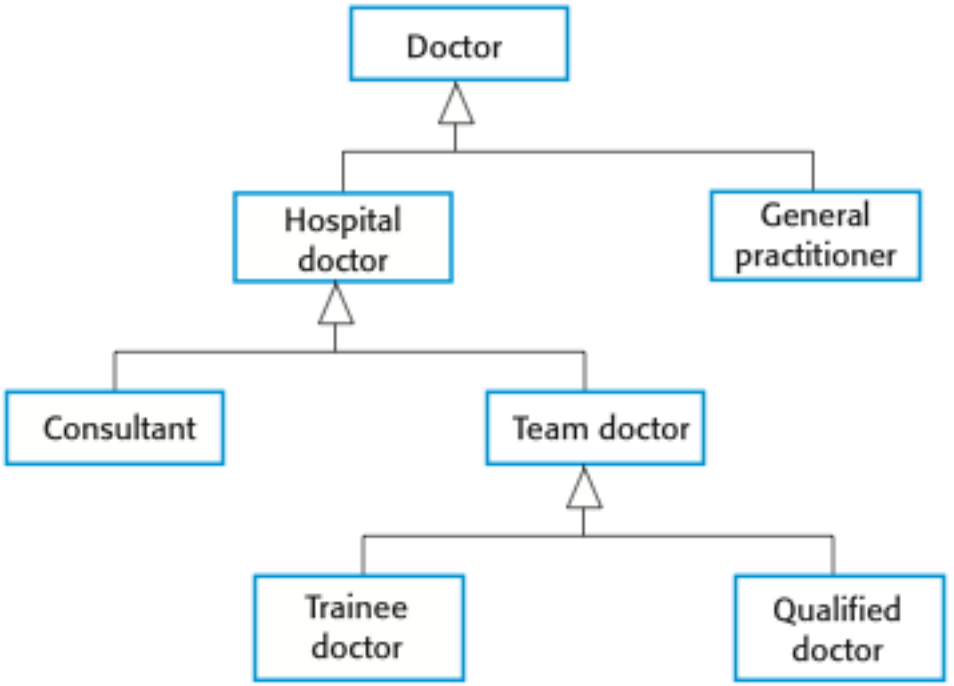
- Class X depends on class Y if X **may need to change** if Y changes.
  - Usually said: “X uses Y”
  - If X knows of Y's existence, then X depends on Y.
  - Shown as a dotted open arrow.
  - Hint: Usually for arguments or local variables.
- Example:

```
class PizzaOrder {
        private List<Pizza> pizzas;
        // ...
        public void slicePizzas() {
                Slicer slicer = new Slicer();
                slicer.slicePizzas(pizzas);
        }
}
```

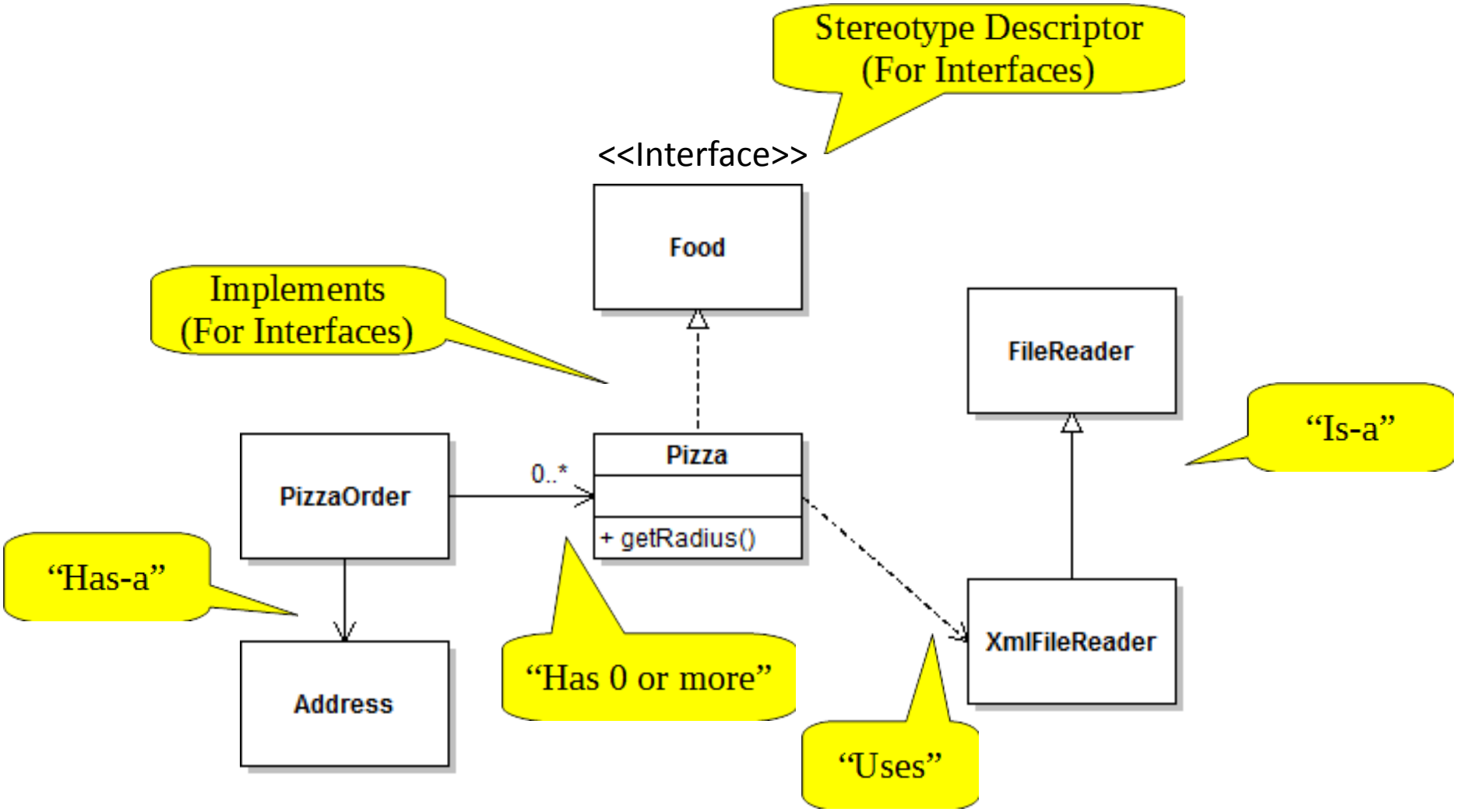


Relationship: Inheritance

- The “Is-A” Relationship:
  - A cell-phone **is a type of** phone: cell-phone inherits from phone.
  - Shown as a hollow arrow pointing from subclass to the superclass (more general class).



Exercise: Label the Relationships



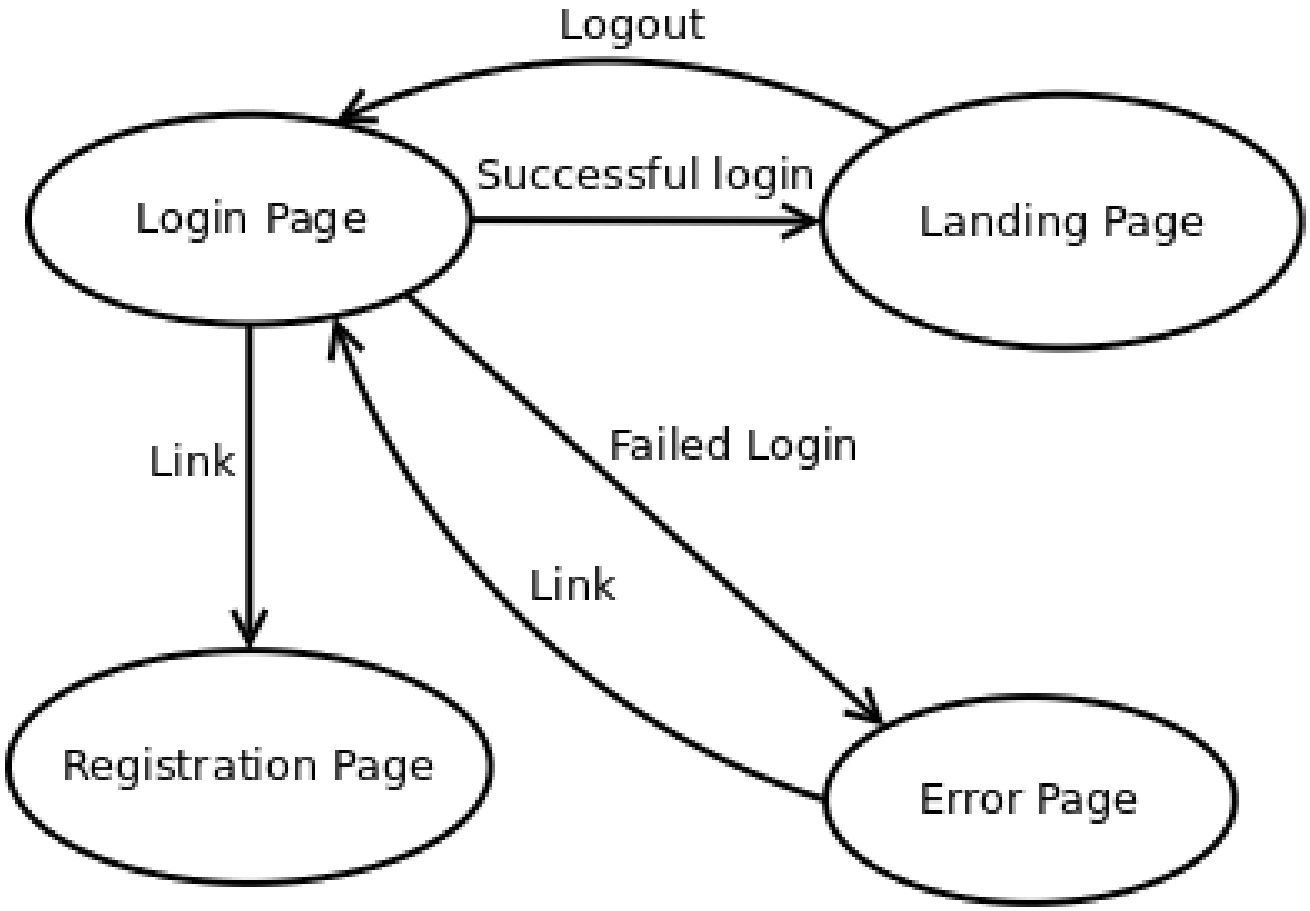
Exercise: UML Class Diagram

```
class Phone {}

class SimCard {}
class SimEjectorTool{}

class Battery {}
class LiPoBattery extends Battery{}
class LithiumIonBattery extends Battery {}

class CellPhone extends Phone{
        private Battery battery;
        private SimCard card;

        void changeSimCard(SimCard card, SimEjectorTool tool) {}
        void setBattery(Battery battery) {}
        int countInstalledApps()
}
```

Behavioural Models

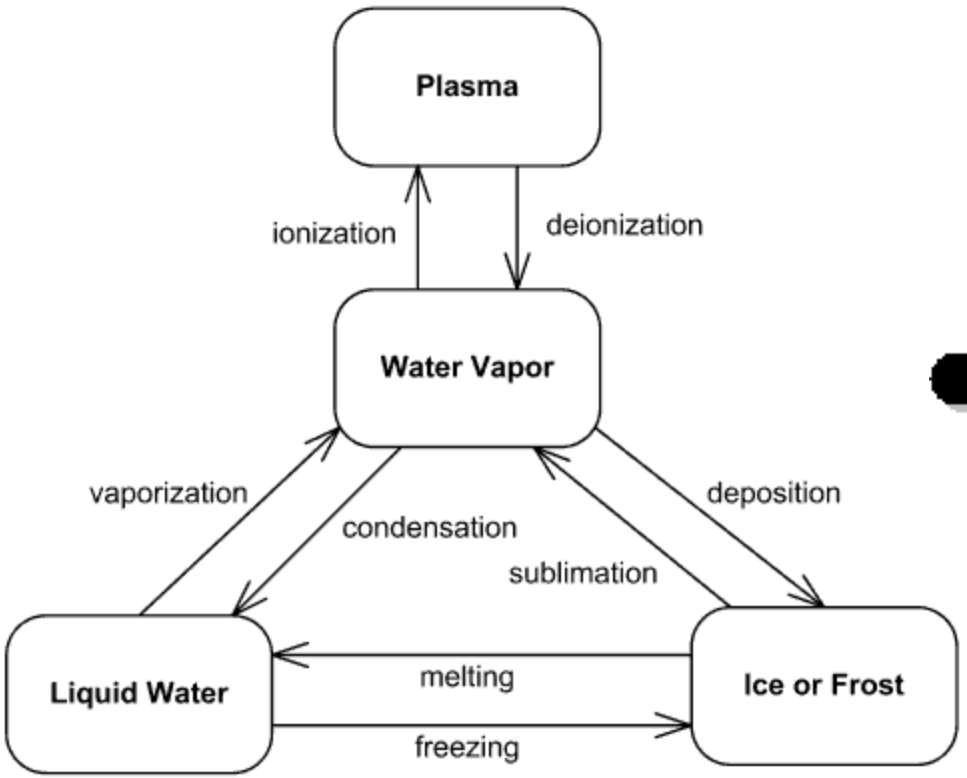
- Models **dynamic behaviour** of a system as it executes.
- Real-time systems are often **event-driven**, with minimal data processing.
  - Ex: microwave oven, alarm clock, etc.
- Event-driven modelling shows how a system responds to **external and internal events**.
  - System has **states**, and events (stimuli) cause state **transitions**.
  - Called a **State Diagram**, or FSM: **Finite State Machine**.

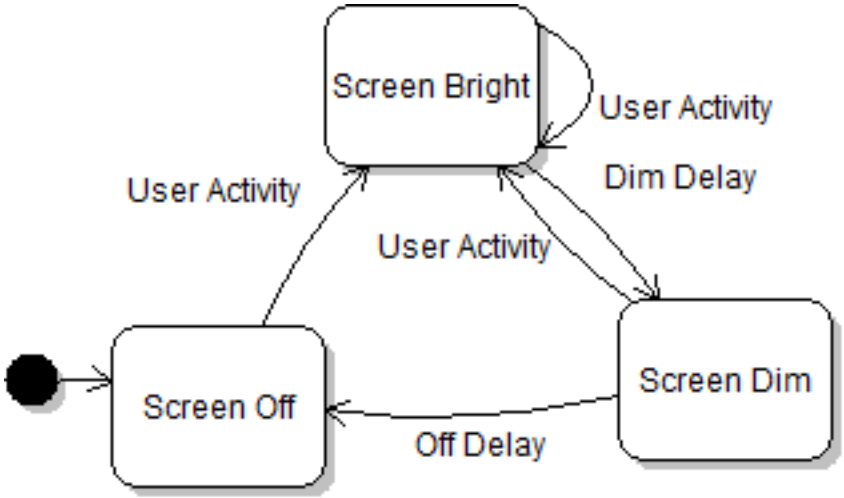
System Authentication Diagram



State Machines

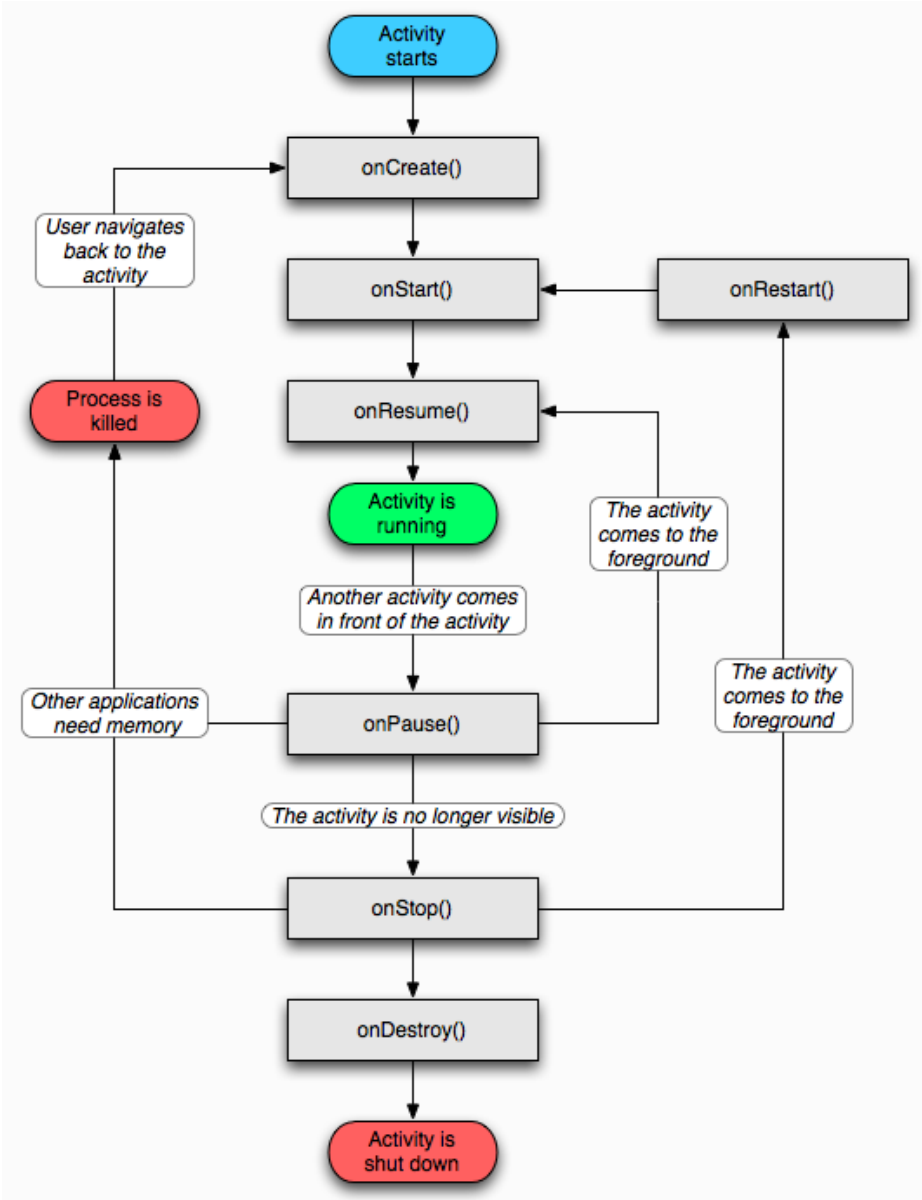
- What are each of the following state machines for?



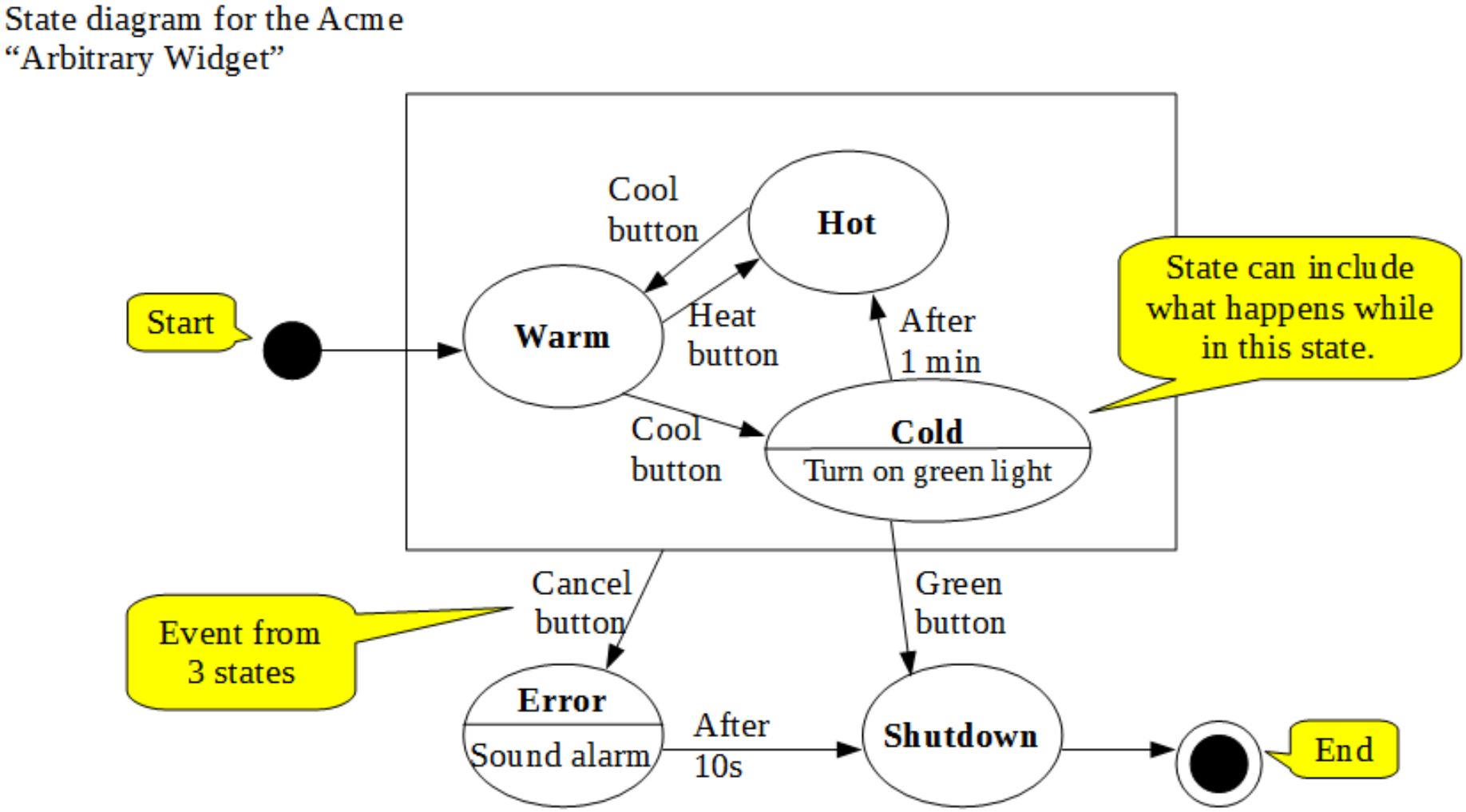


Android

- Many events can occur in the lifetime of an Android activity.
- Trace the following:
  - Creation
  - While running, switch to home screen.
  - While in background, killed by OS.



UML State Diagram Components



Example: Boss Fight State Diagram

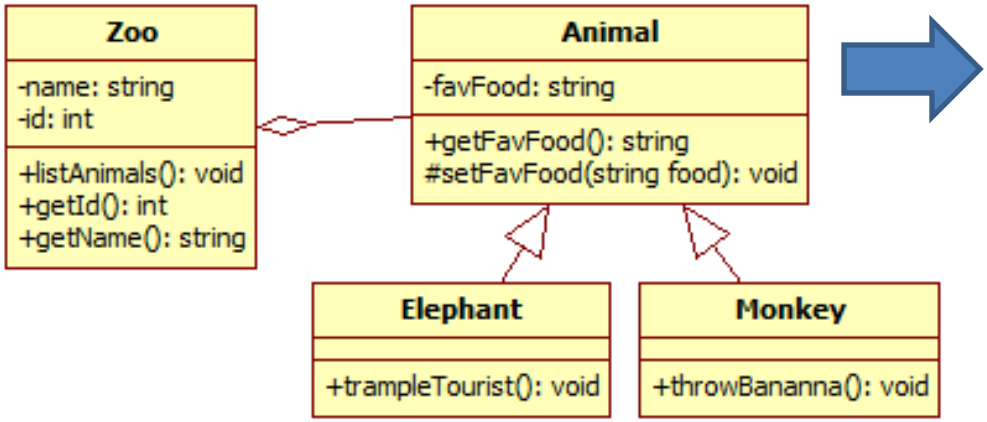
- Imagine you are in a game battling an epic dragon. Draw a state diagram for the “Boss”.
  - **Ground Phase**: Dragon on ground (start).
    - After 1 minute goes to air phase.
  - **Air Phase**: Dragon in air, summons a minion.
    - After minion is killed, go to ground phase.
  - **Burn Phase**:
    - When boss’s health reaches 30% he lands and starts breathing fire.
  - **Tamed**: Boss at 0% health, players have tamed the dragon.
  - **Enraged**:
    - After 5 minutes, dragon heals fully, takes to the air and enrages killing everyone.
  - **Boss Win**: If all players die.

Model-Driven Engineering

- An approach to software development where models rather than programs are the principal outputs of the development process.
  - Programs automatically generated from the models.
- **Pros**
  - Work at a higher levels of abstraction.
  - Cheaper port to new platforms: code is generated!
- **Cons**
  - Models for abstraction not always suited to implementation.
  - Still somewhat theoretical, not well supported.

Model-Driven Engineering Example

- **StarUML** Generates C++ code from class diagram
  - Generates all .h files and function stubs in .cpp files.
- **Umple** is for Java.




```
//  Generated by StarUML(tm) C++ Add-In
//
//  @ Project : Untitled
//  @ File Name : Zoo.h
//  @ Date : 20/02/2014
//  @ Author :

#if !defined(_ZOO_H)
#define _ZOO_H

class Zoo {
public:
        void listAnimals();
        int getId();
        string getName();
private:
        string name;
        int id;
};

#endif  //_ZOO_H
```

Recap – A Model of Brevity

- **Model**: Abstract view of system; ignores some details
- **System's Context**
  - Context models show environment around system
- **Interactions**
  - Use cases - external actor interactions with system
- **Structural Models**: Show system architecture
  - Class Diagrams shows static structure of classes
- **Behavioural Models**: Dynamic behaviour of executing system.
  - State Diagram - States and internal/external events
- **Model-Driven Engineering**: Build the model, and then tools automatically transformed to executable code.