Project Dilverable 1 Tips

Listed below are some key features to include in the appropriate diagrams as you work on Project Dilverable 1

1. Use-Case Diagram
    a. Don't forget to include <<extends>> and <<includes>> relationships between use cases where relevant
    b. Pick 2 use cases from your diagram and provide the complete textual description.

2. Class Diagram
    a. Include Instance variables and methods of classes (check example below)
    b. Provide relationships and associations between classes
    c. Provide multiplicity
    d. Indicate Aggrigation, Composition or inheritance

3. Activity Diagram
    a. Use split and synchronize constructs
    b. Oragnize Activity into sutable lanes

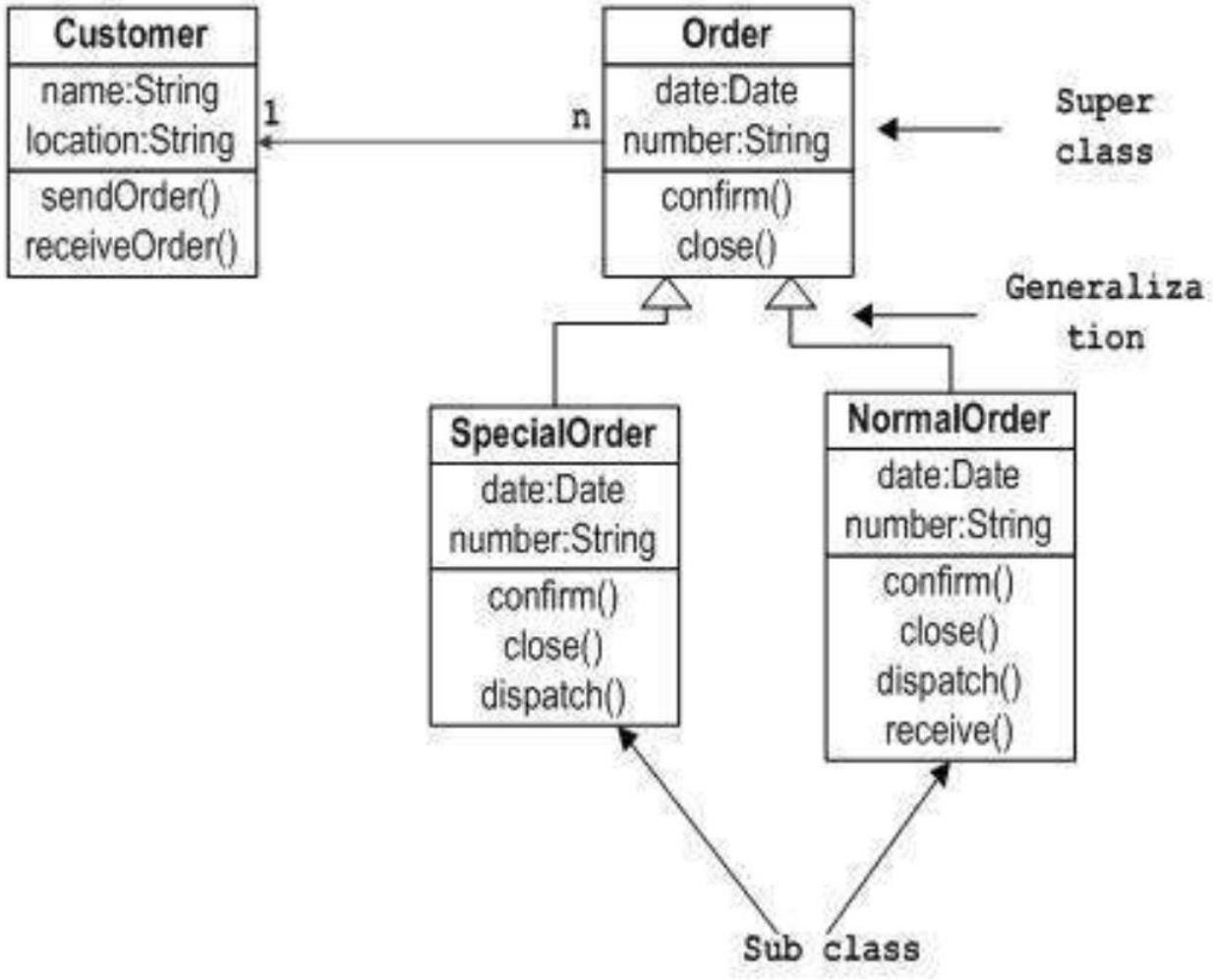
4. State Diagram
    a. Identify all states of a Connect4 Game object
    b. Incldue Transitions between states labled by triggers
    c. Include start and end state

Listed below are some some refreshers and examples on the topics covered in Project Dilverable 1

Class Diagram – used to represent the structure of the system used

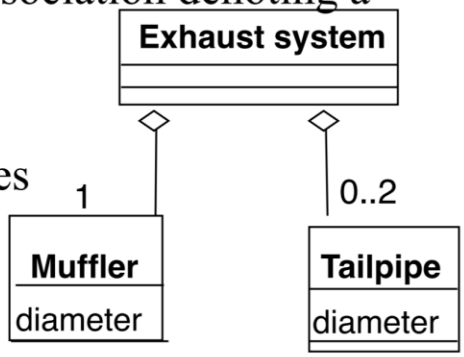
- Class name
- Attributes (variables)
- Method names

Sample Class Diagram

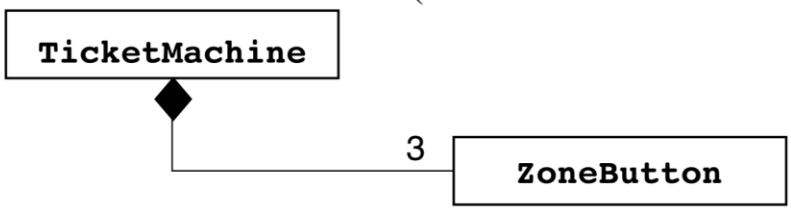


Aggregation

- An *aggregation* is a special case of association denoting a "consists-of" hierarchy
- The *aggregate* is the parent class, the components are the children classes



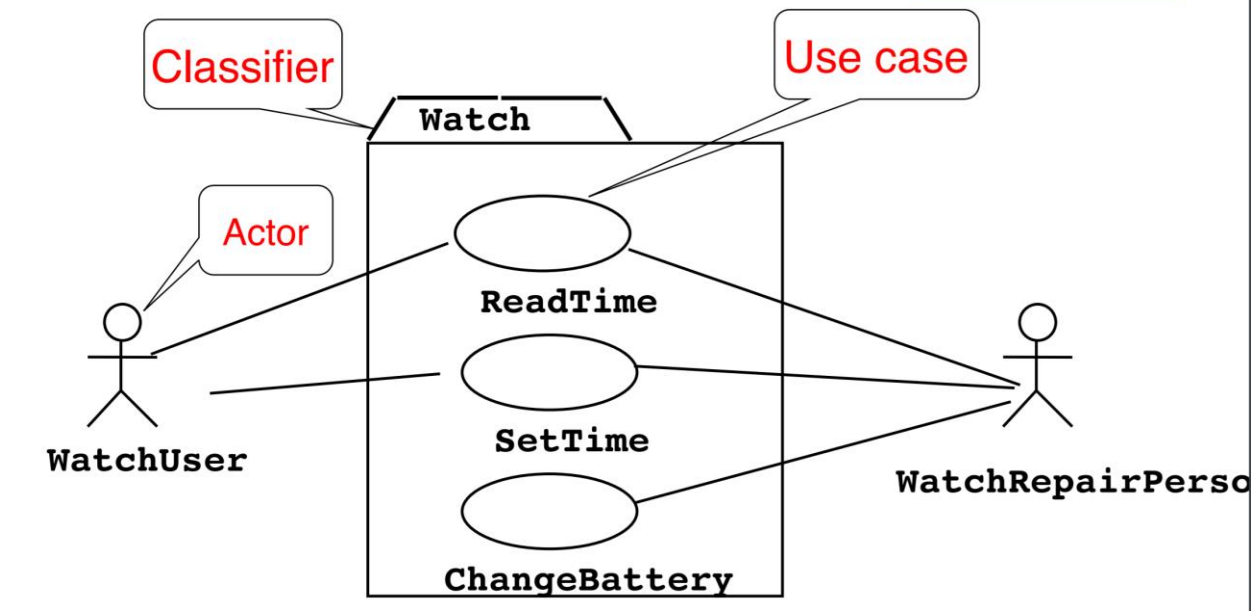
A solid diamond denotes *composition*: A strong form of aggregation where the *life time of the component instances* is controlled by the aggregate. That is, the parts don't exist on their own ("the whole controls/destroys the parts")

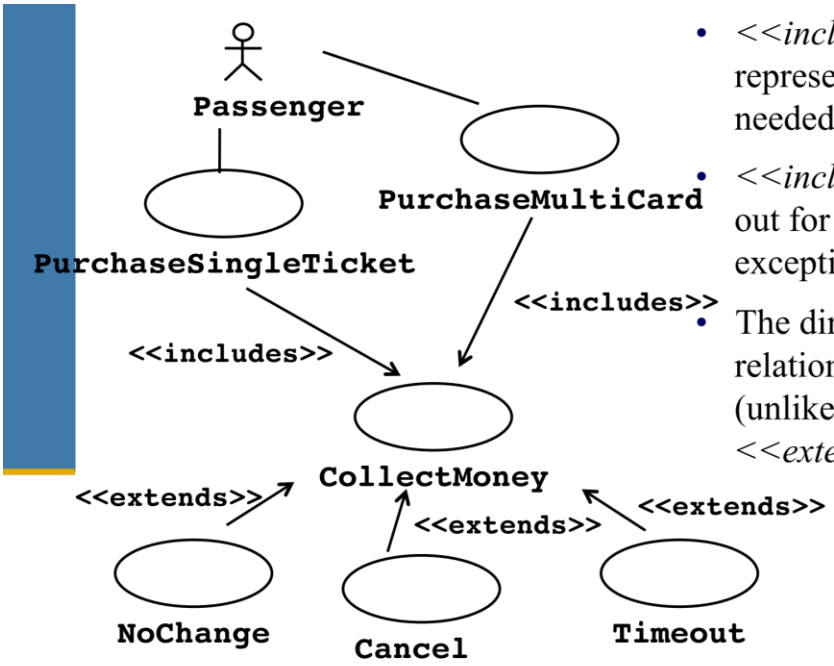


Use Case – represents functionality provided by the system

- Actors - represents a user of the system
- Classifier – Describes what it is you're modeling. Usually the class name.
- Use case – Different functionalities provided by the class (your different methods)

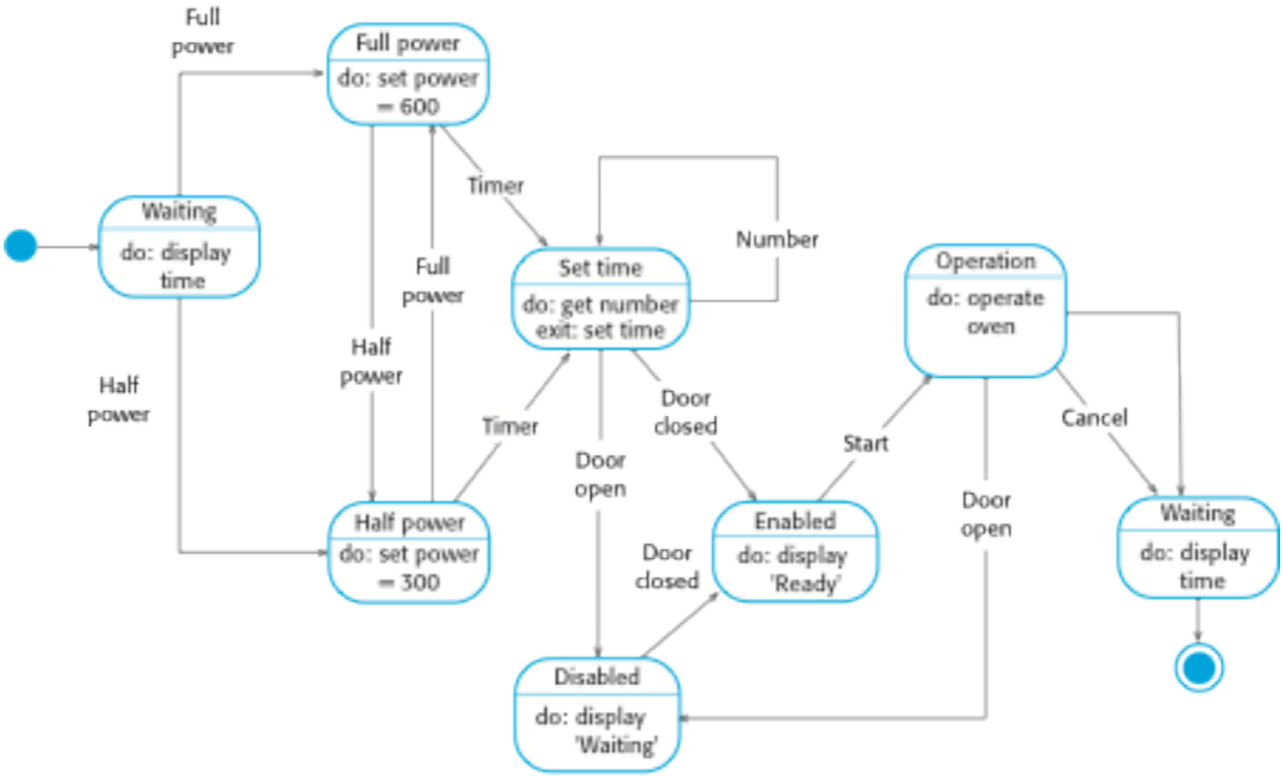
Exercise: Use Case Diagram





- <<*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).

State Diagrams - Model the behavior of the system in response to external and internal events.



State – current state of the system (waiting, full power, half power, set time, etc…)

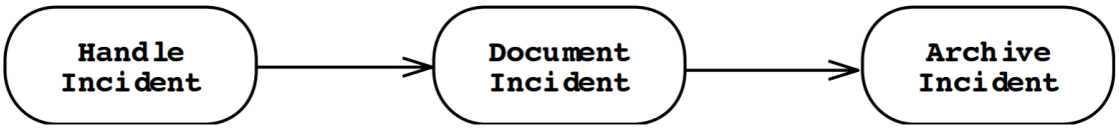
Transition – Arrows leading from one state to another

Event – What is causing the system to change states Full power -> (Waiting, Full power)

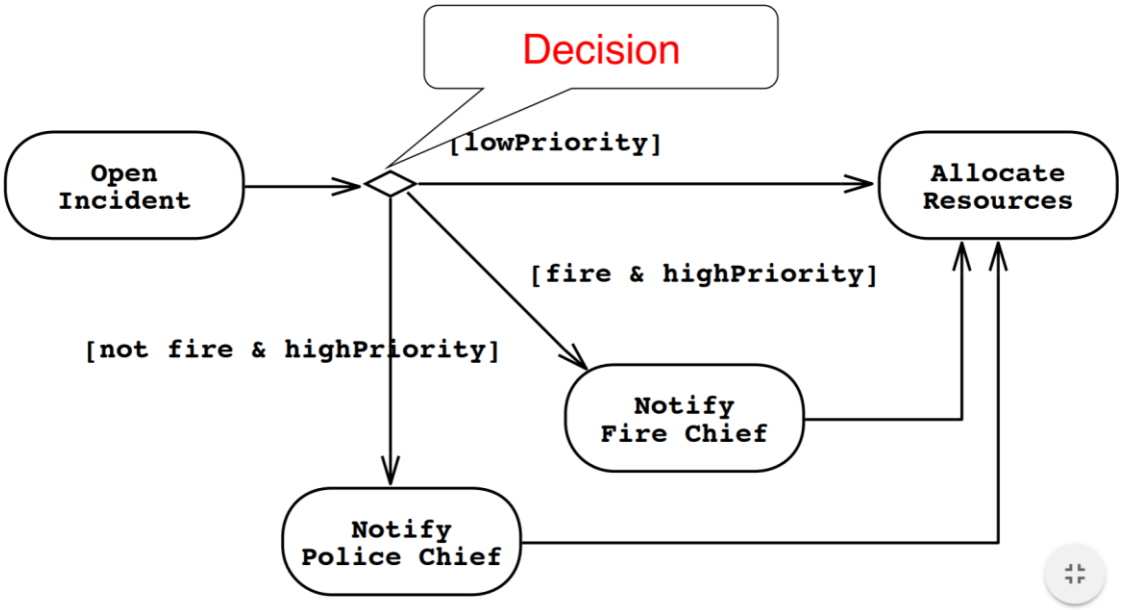
Activity Diagram-

Activity Diagrams

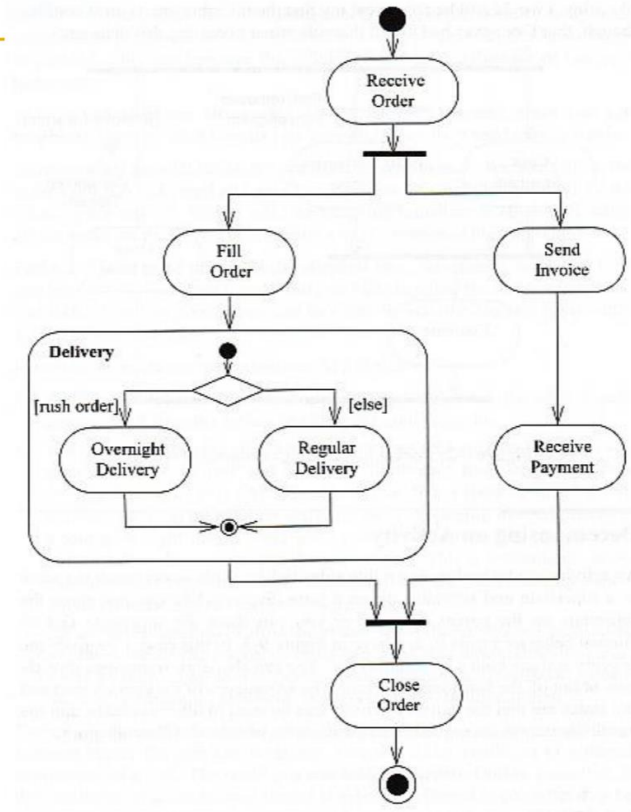
- An activity diagram is a special case of a state chart diagram
- The states are activities (“functions”)
- An activity diagram is useful to depict the workflow in a system



Activity Diagrams allow to model Decisions



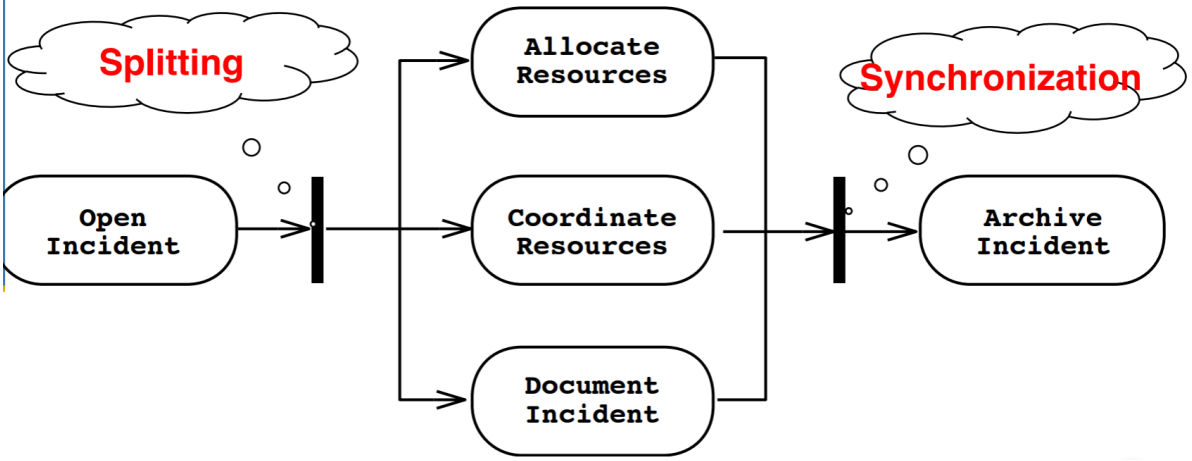
Exercise: Activity Diagram



- We concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: class diagram
  - Dynamic model: sequence diagrams, statechart and activity diagrams

Activity Diagrams can model Concurrency

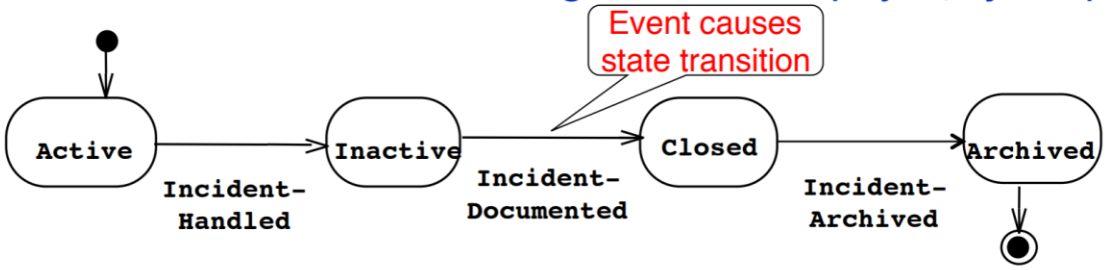
- Synchronization of multiple activities
- Splitting the flow of control into multiple threads



Activity Diagram vs. Statechart Diagram

Statechart Diagram for Incident

Focus on the set of attributes of a single abstraction (object, system)



Activity Diagram for Incident

(Focus on dataflow in a system)