



UML Models

Overview

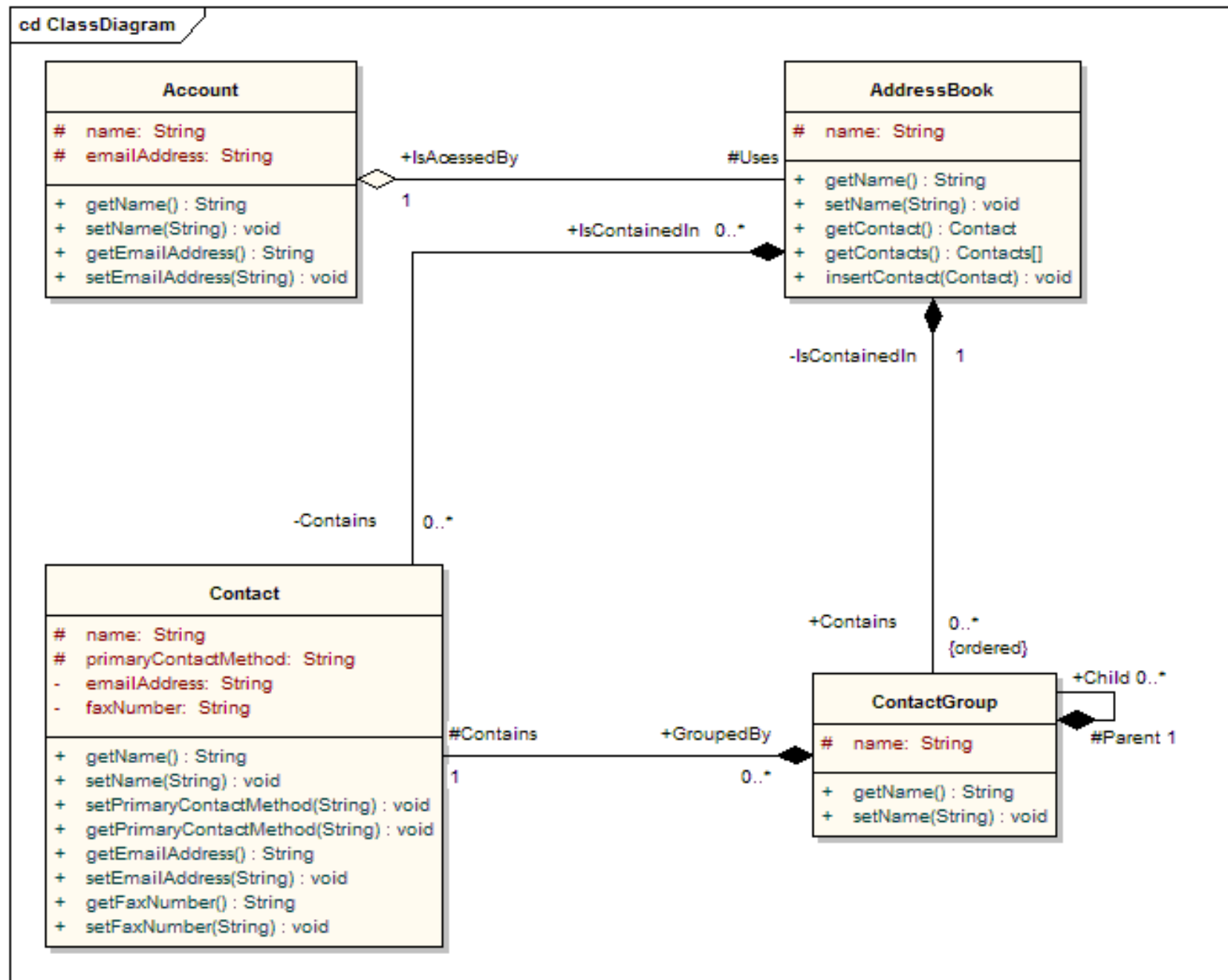
- Class diagrams
- Statechart diagrams
- Activity diagrams



Class Diagrams

Class Diagrams

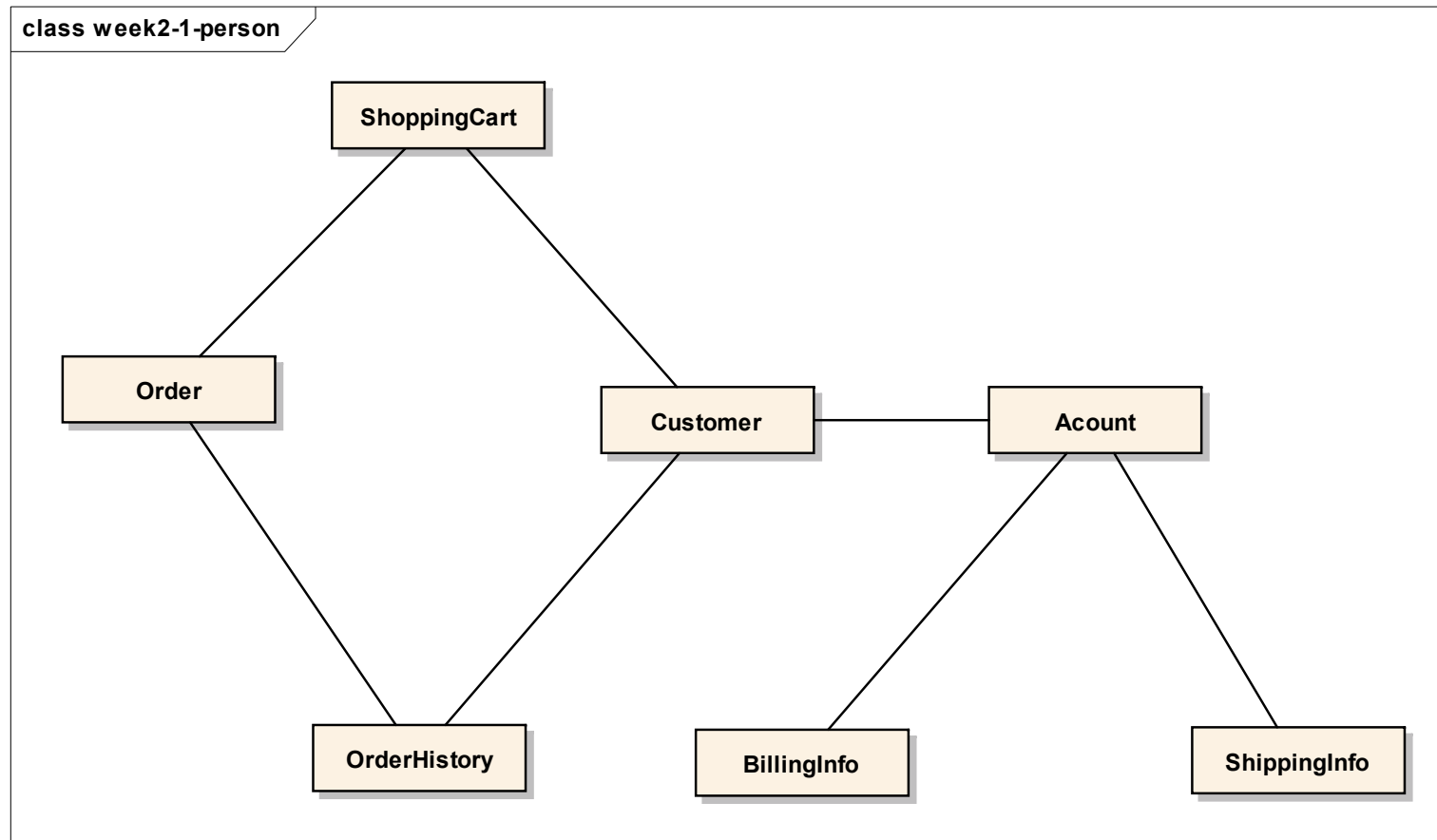
- represents the logical structure of the system
 - based on the classes and things that make up the model
- a static model describing what exists in the system and what attributes and behavior it has
 - does not focus on how something is done
- class diagrams are most useful to illustrate relationships
 - generalizations, aggregations, and associations are used in reflecting inheritance, composition or usage, and connections, respectively



Class Diagrams: Example

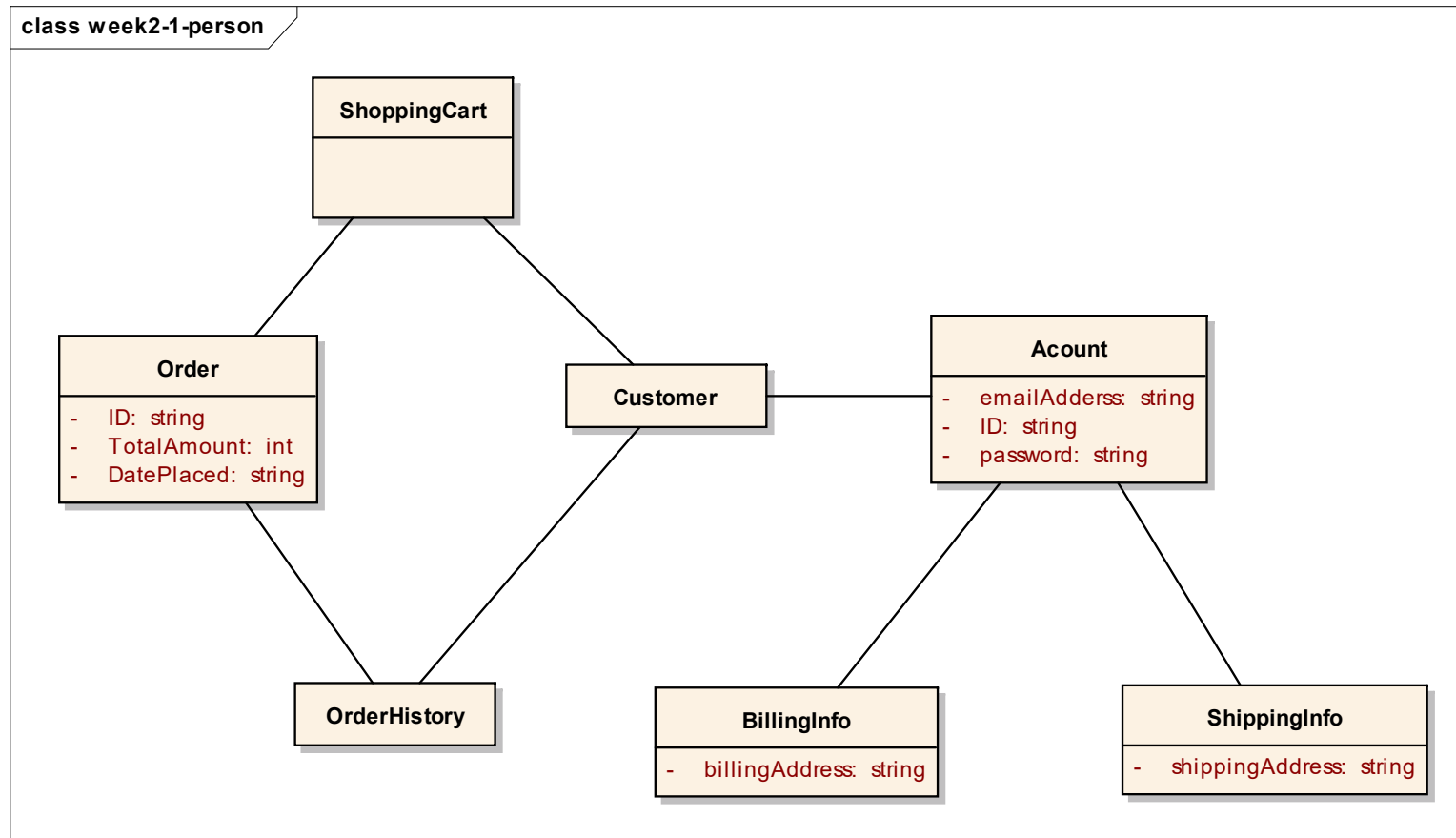
Domain-Level Class Diagrams

- Just show the names of classes
- Show part of the initial core vocabulary with which system modeling can proceed

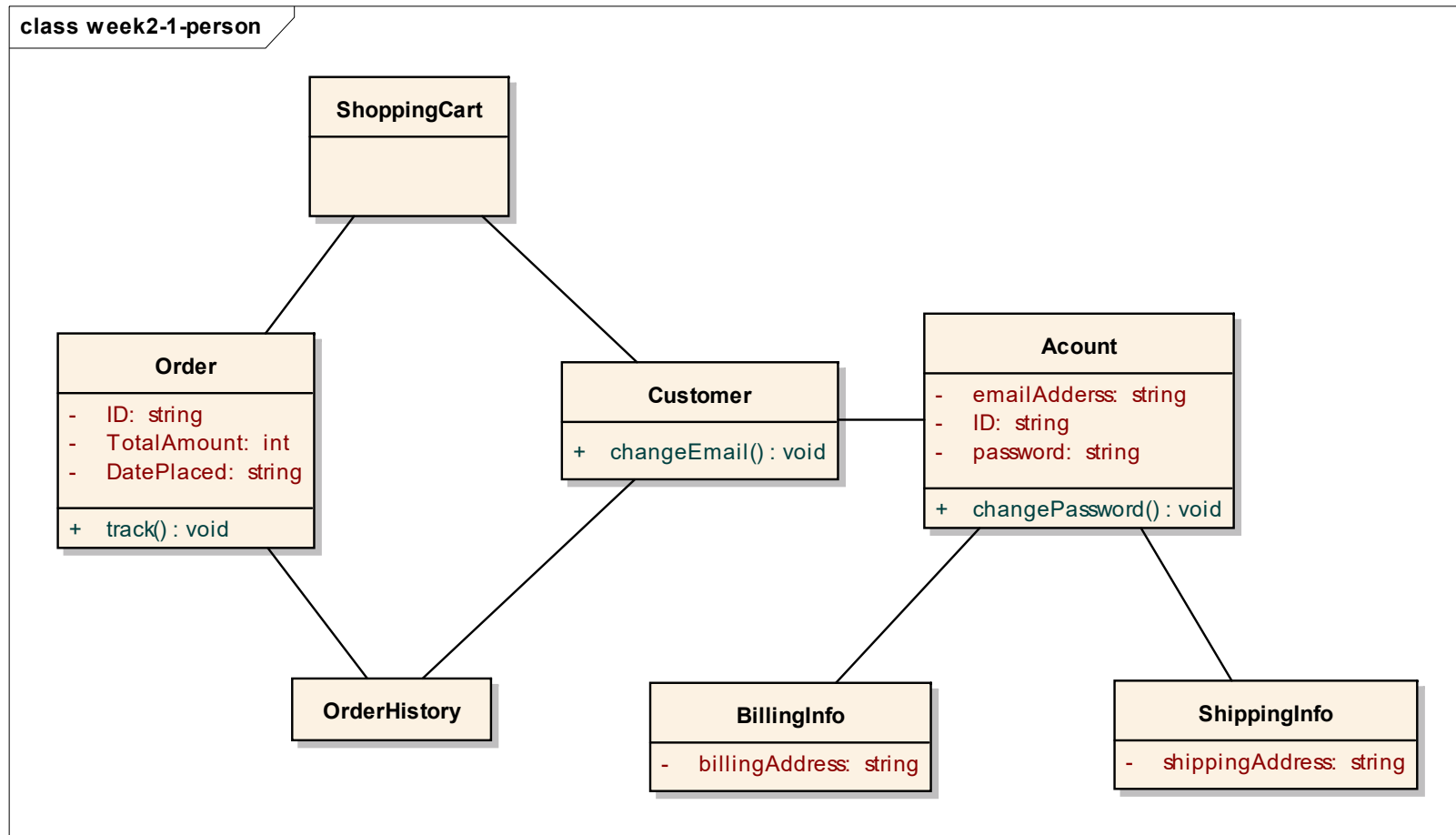


Analysis-Level Class Diagrams

- Show attributes, some operations
- Show part of the initial core vocabulary with which system modeling can proceed



Design-Level Class Diagrams



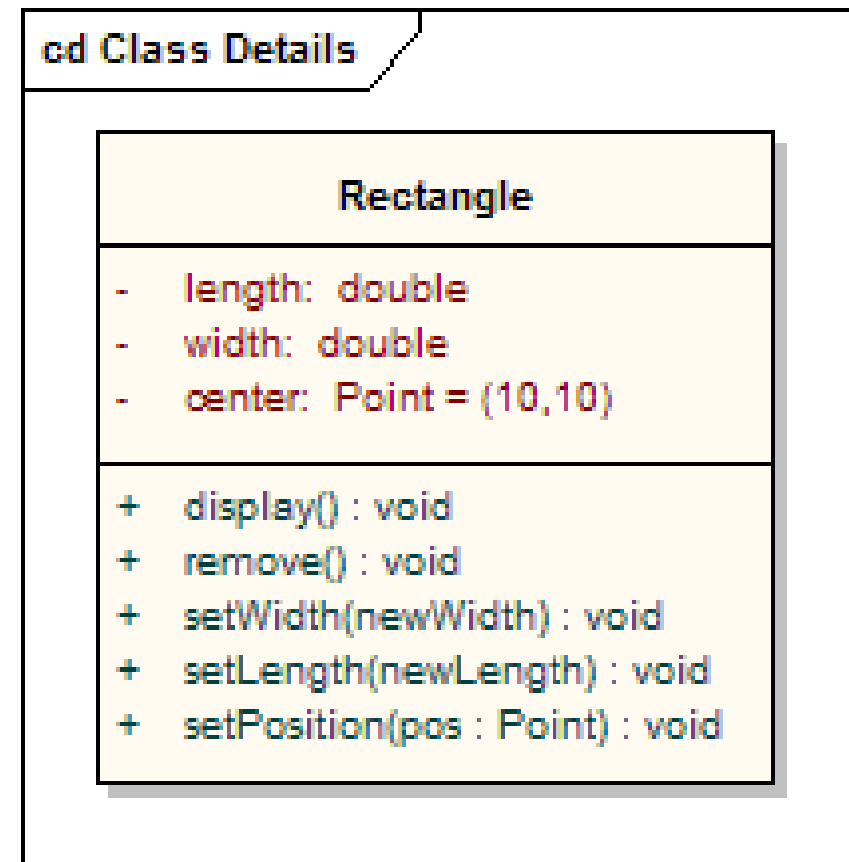
Classes and Objects

- Class: A class is a collection of things or concepts that have the same characteristics
- An Object that belongs to a particular class is often referred to as an *Instance* of that class

Class Diagrams:

Class Visibility

- classes are represented by rectangles which show the name of the class and optionally the name of the operations and attributes
 - compartments are used to divide the class name, attributes and operations
- visibility of the element
 - + used for public visibility
 - - used for private visibility
 - # used for protected visibility

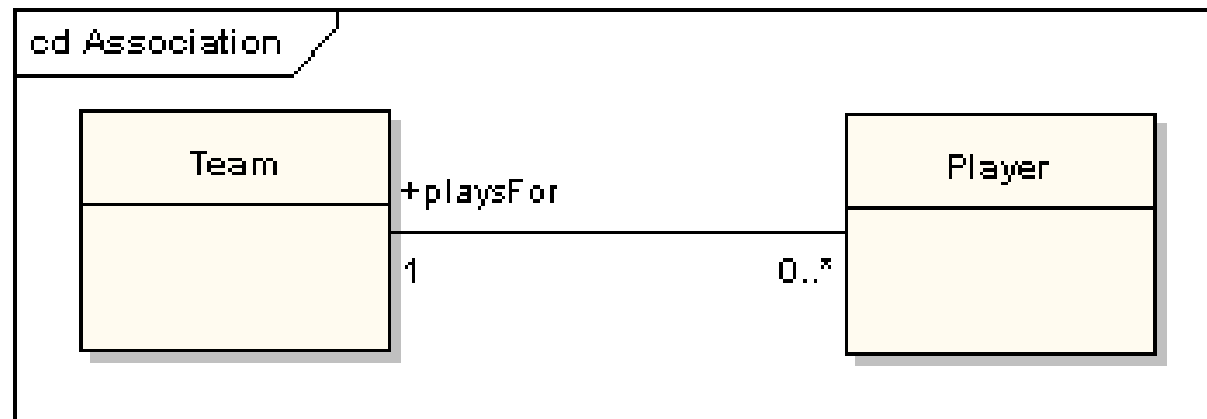


Class Diagrams: Associations

- an association line connecting two class diagrams indicates that there is a linkage between the classes
- association is the general relationship type between elements
- association connector may include
 - named roles at each end
 - cardinality and multiplicity
 - direction
 - constraints

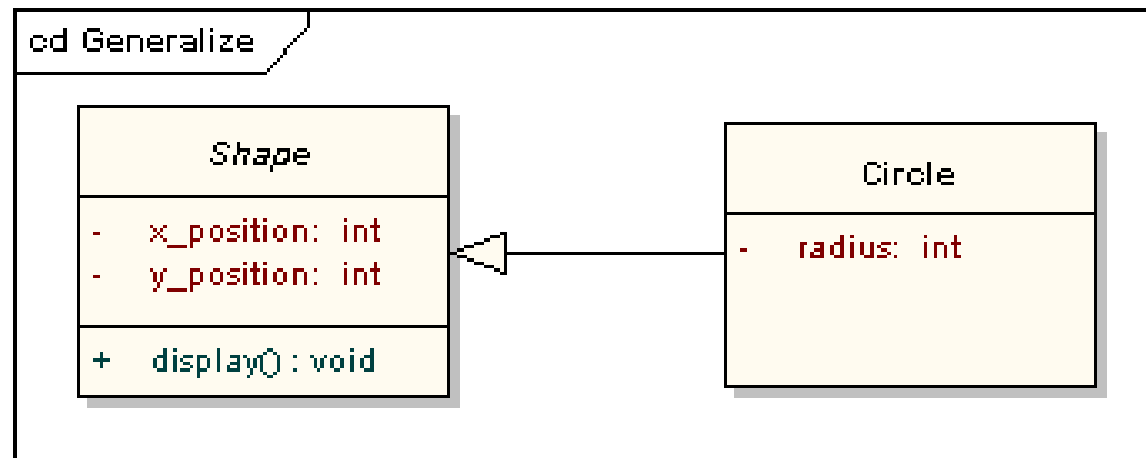
Class Diagrams: Associations ...

- when code is generated for class diagrams, named association ends become instance variables in the target class
- for the example below, "playsFor" will become an instance variable in the "Player" class



Class Diagrams: Generalizations

- used to indicate inheritance – drawn from the specific classifier to a general classifier, the generalize implication is that the source inherits the target's characteristics
- example shows a parent class generalizing a child class
 - implicitly, an instantiated object of the Circle class will have attributes `x_position`, `y_position` and `radius` and a method `display()`.
 - note that the class "Shape" is abstract, shown by the name being italicized

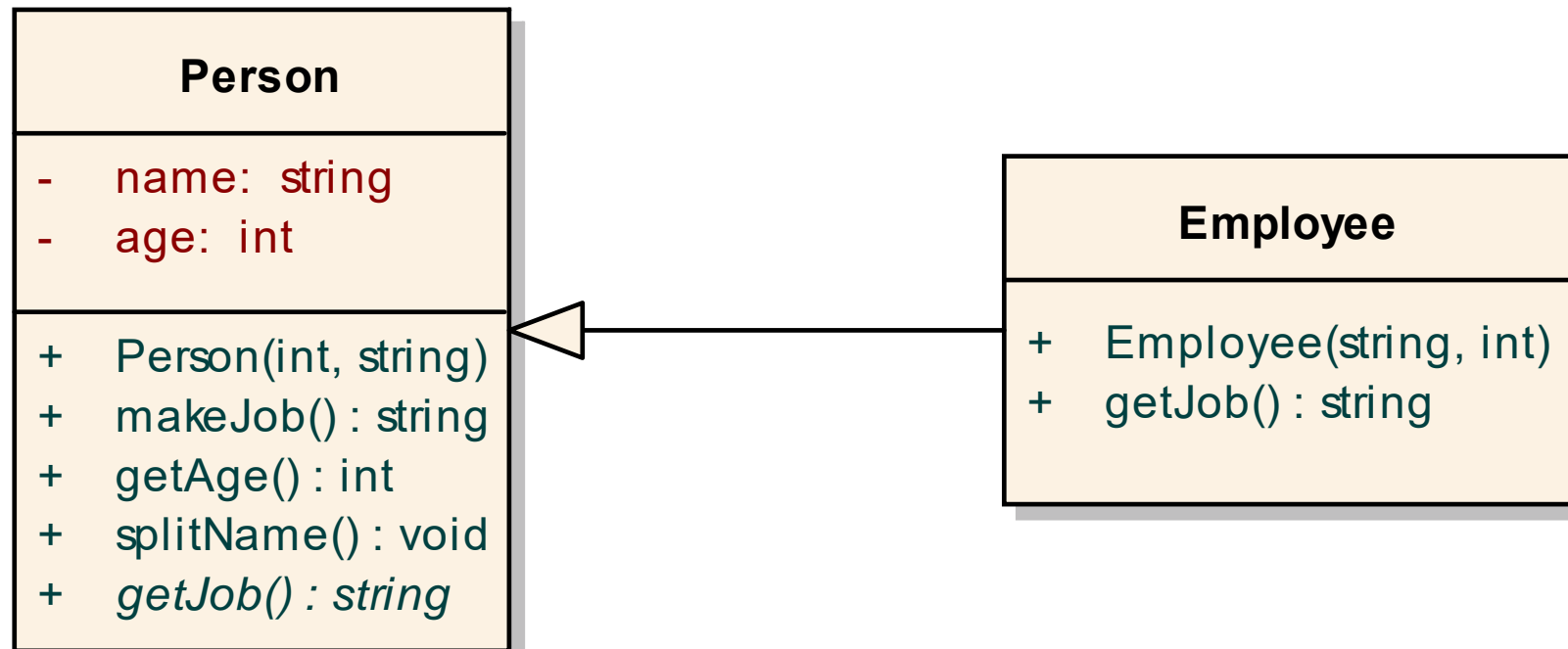


Inheritance

- A class **inherits** from another class if it receives some or all of the qualities (e.g., methods) of that class
 - **Starting class**: base, super, parent, generalized class
 - **Inheriting class**: derived, sub, child, specialized class

Inheritance Example

class week2-1-person



Inheritance Example...

■ C#

```
public class Person {  
    private string name;  
    private int age;  
    public Person(int ag, string nm){  
        name=nm;  
        age=ag;  
    }  
    public string makeJob(){  
        return "hired";  
    }  
    public int getAge(){  
        return age;  
    }  
  
    public void splitName()  
    {  
    }  
    public abstract string getJob();  
}
```

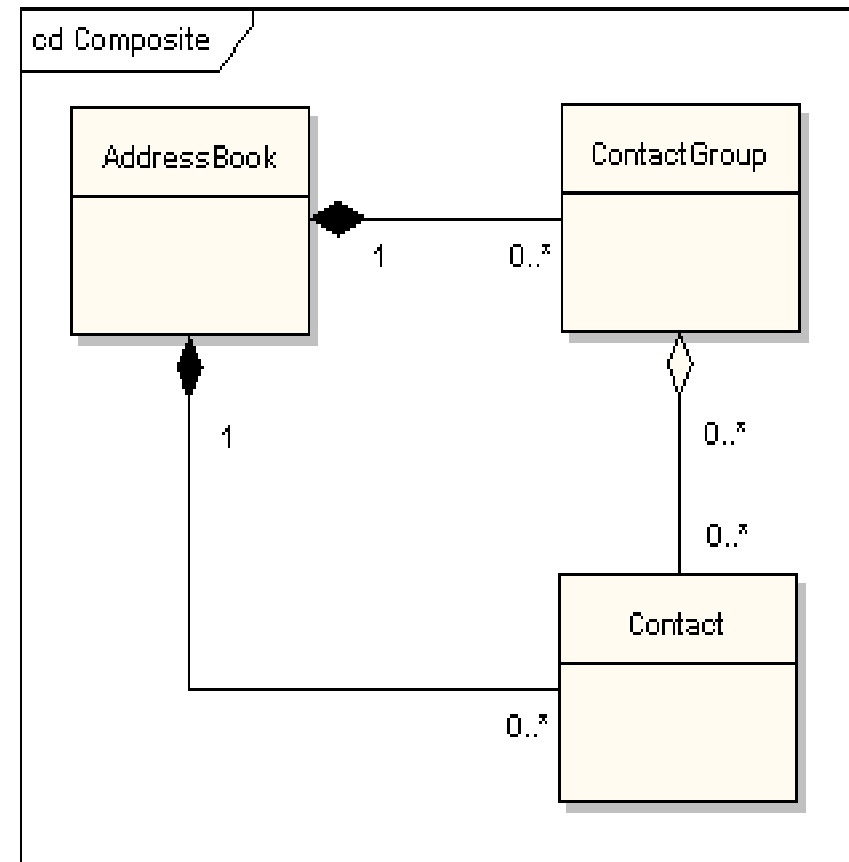
```
public class Employee : Person{  
    public Employee(string nm, int ag){  
    }  
    public override string getJob(){  
        return "Worker";  
    }  
}
```


Class Diagrams: Aggregations

- aggregations are used to depict elements which are made up of smaller components – shown by a white diamond-shaped arrowhead pointing towards the target or parent class
- stronger form of aggregation - a composite aggregation - is shown by a black diamond-shaped arrowhead and is used where components can be included in a maximum of one composition at a time
- if the parent of a composite aggregation is deleted, usually all of its parts are deleted with it
 - however a part can be individually removed from a composition without having to delete the entire composition
 - compositions are transitive, asymmetric relationships and can be recursive

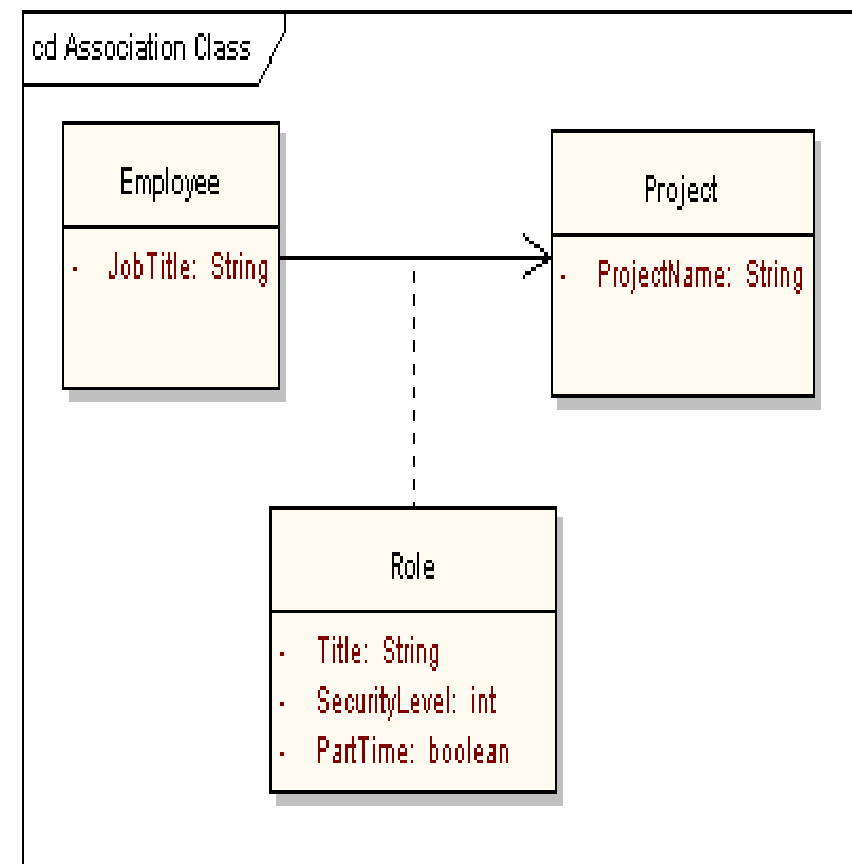
Class Diagrams: Aggregations ...

- an address book is made up of a multiplicity of contacts and contact groups
- a contact group is a virtual grouping of contacts; a contact may be included in more than one contact group
- if you delete an address book, all the contacts and contact groups will be deleted too
- if you delete a contact group, no contacts will be deleted



Class Diagrams: Association Class

- a construct that allows an association connection to have operations and attributes
- example shows that the role the employee takes up on the project is a complex entity in its own right and contains detail that does not belong in the employee or project class
- e.g., an employee may be working on several projects at the same time and have different job titles and security levels on each.



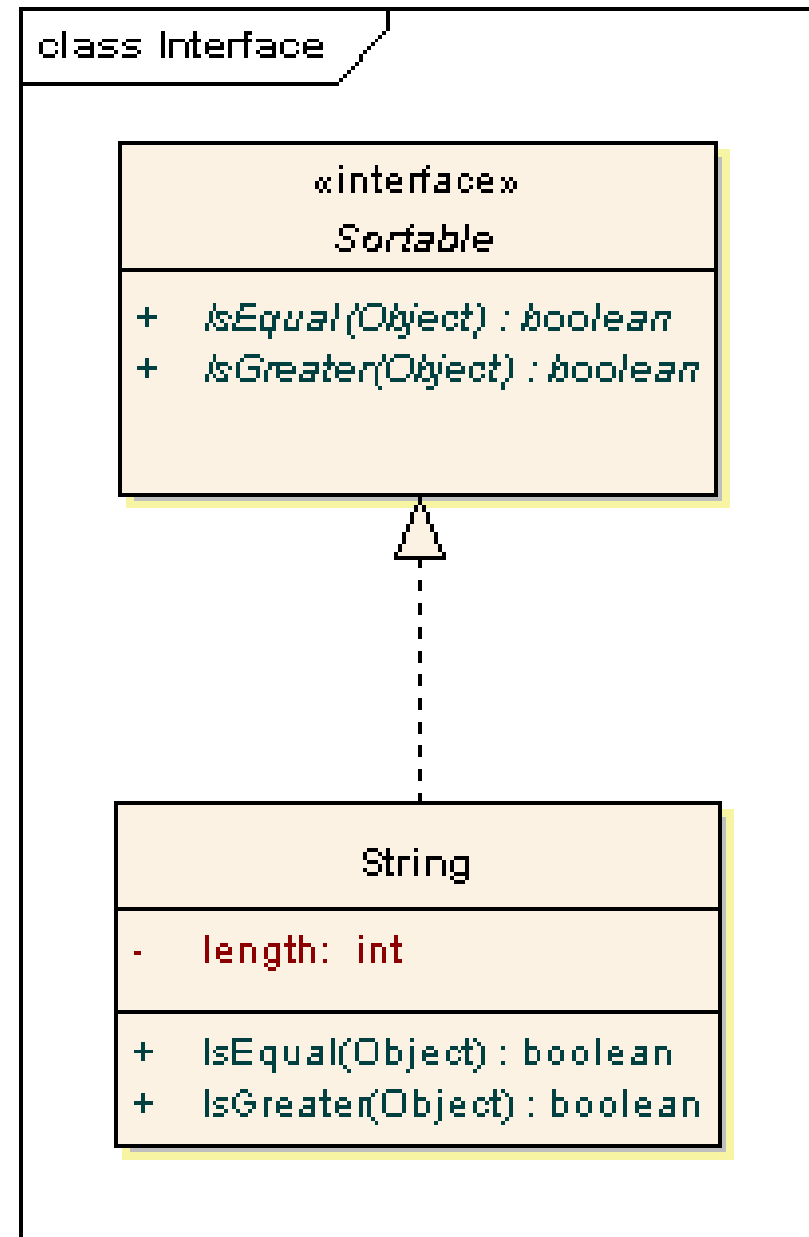
Class Diagrams: Dependency

- a dependency is used to model a wide range of dependent relationships between model elements
- it would normally be used early in the design process where it is known that there is some kind of link between two elements, but it is too early to know exactly what the relationship is
- later in the design process, dependencies will be stereotyped, or replaced with a more specific type of connector
 - stereotypes available include «instantiate», «trace», «import», and others

Class Diagrams: Realizations

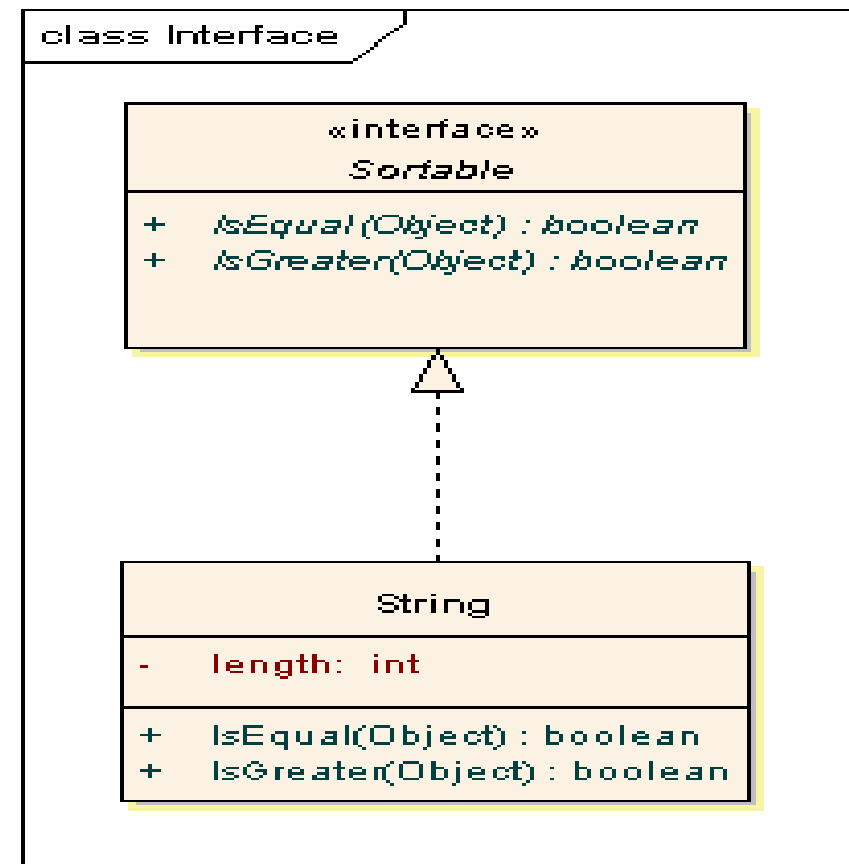
- the source object implements or realizes the destination
- realizations are used to express traceability and completeness in the model
 - a business process or requirement is realized by one or more use cases, which are in turn realized by some classes, which in turn are realized by a component, etc.
- mapping requirements, classes, etc. across the design of your system, up through the levels of modeling abstraction, ensures the big picture of your system remembers and reflects all the little pictures and details that constrain and define it
- a realization is shown as a dashed line with a solid arrowhead

Class Diagrams: Realizations ...

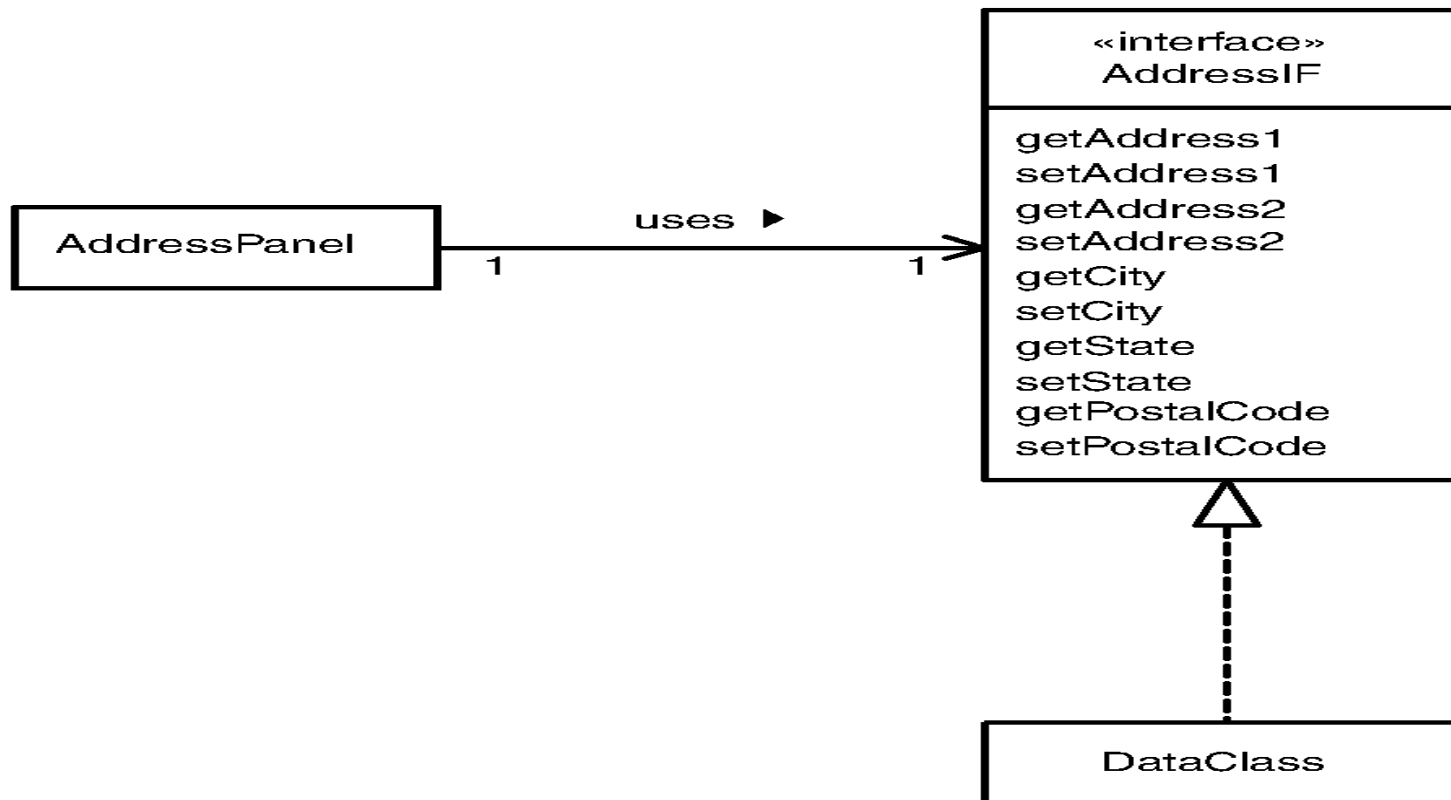


Class Diagrams: Interfaces

- an interface is a specification of behavior that implementers agree to meet; it is a contract
- by realizing an interface, classes are guaranteed to support a required behavior, which allows the system to treat non-related elements in the same way – that is, through the common interface



Class Interface Example



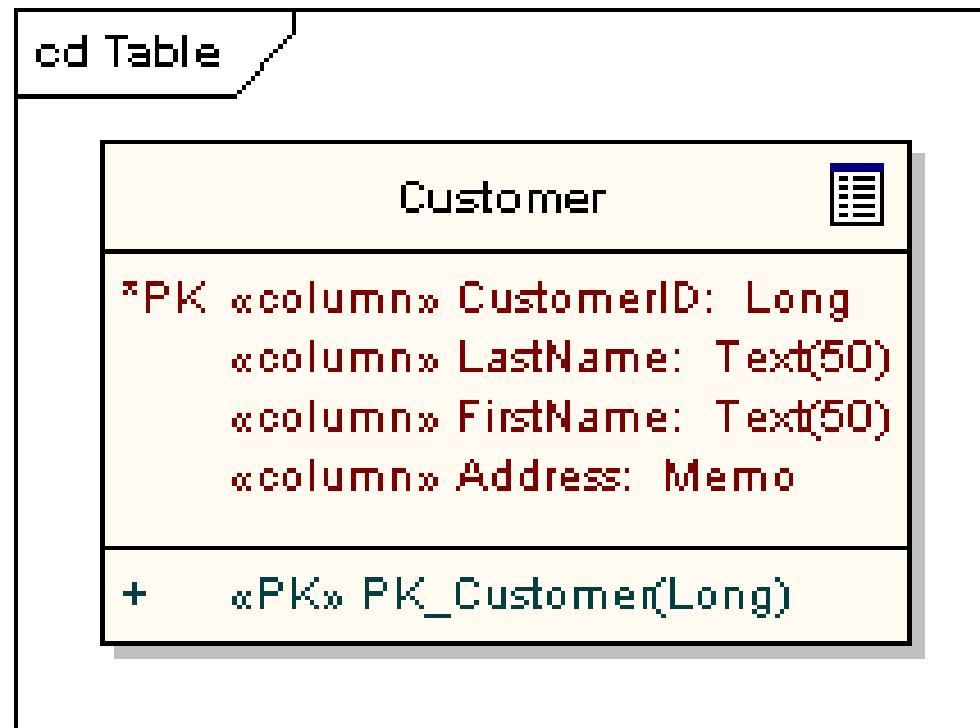

```
public interface AddressIF {  
    public String getAddress1();  
    public void setAddress1(String address1);  
    ...  
    public String getPostalCode() ;  
    public void setPostalCode(String PostalCode);  
} // interface AddressIF
```

```
class ReceivingLocation extends Facility implements AddressIF{  
    private String address1;  
    ...  
    private String postalCode;  
    ...  
    public String getAddress1() { return address1; }  
    public void setAddress1(String address1) {  
        this.address1 = address1;  
    } // setAddress1(String)  
    ...  
    public String getPostalCode() { return postalCode; }  
    public void setPostalCode(String postalCode) {  
        this.postalCode = postalCode;  
    } // setPostalCode(String)  
} // class ReceivingLocation
```

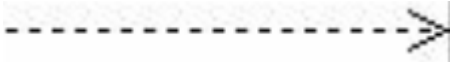
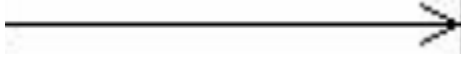


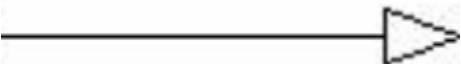
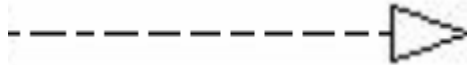
Class Diagrams: Tables

- a table is an example of what can be done with stereotypes, and is drawn with a small table icon in the upper right corner
- table attributes are stereotyped «column»
- most tables will have a primary key, being one or more fields that form a unique combination used to access the table, plus a primary key operation which is stereotyped «PK»
- some tables will have one or more foreign keys, being one or more fields that together map onto a primary key in a related table, plus a foreign key operation which is stereotyped «FK»

Class Diagrams: Tables ...



Class Diagram Relationship Summary

<u>Relationship</u>	<u><any line></u>	<u>meaning</u>
Dependency		"uses a"
Association		"has a"
Aggregation		"owns a"
Composition		"is composed of"
Generalization		"is a"
Realization		"implements"

Identifying Classes

<i>Part of speech</i>	<i>Model component</i>	<i>Example</i>
Proper noun	object	Jim Smith
Common noun	class	Toy, doll
Doing verb	method	Buy, recommend
being verb	inheritance	is-a (kind-of)
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	3 years old
transitive verb	method	enter
intransitive verb	method (event)	depends on

Example

Flow of events:

- The customer enters the store to buy a toy.
- It has to be a toy that his daughter likes and it must cost less than \$50.
- He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

**Is this a good use
Case?**

Not quite!

An assistant helps him.

The suitability of the game depends on the age of the child.

His daughter is only 3 years old.

The assistant recommends another type of toy, namely the boardgame “Monopoly”.

**“Monopoly” is probably a
left over from the scenario**

**The use case should
terminate with the
customer leaving the store**

Textual Analysis using Abbot's technique

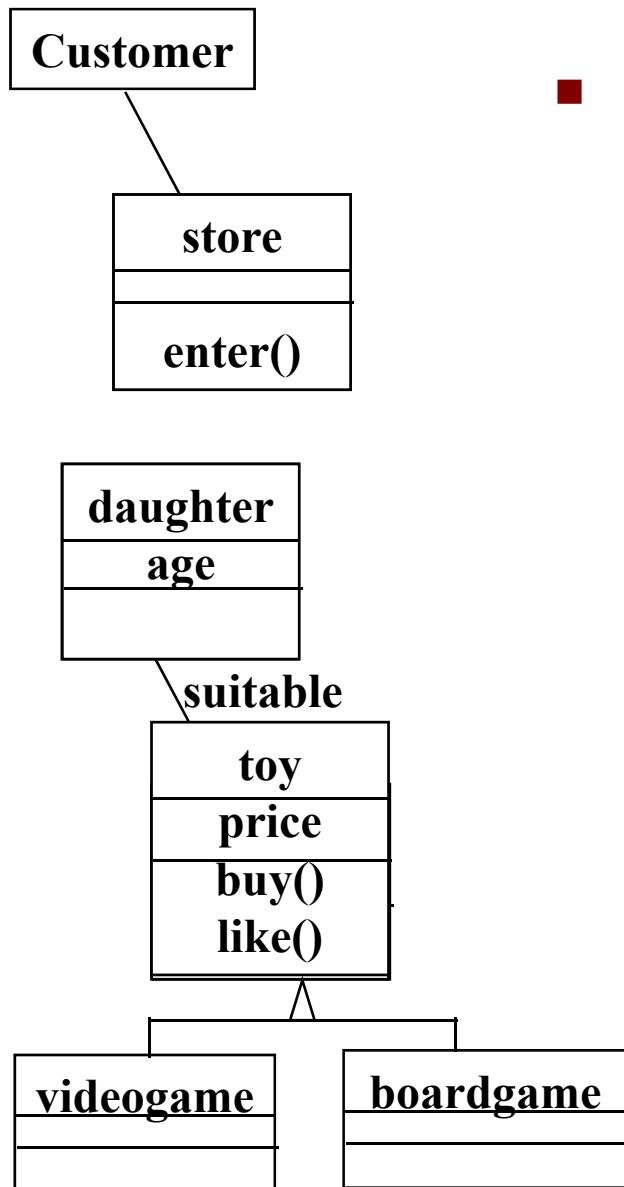
<i>Example</i>	<i>Grammatical construct</i>	<i>UML Component</i>
"Monopoly"	Concrete Thing	Object
"toy"	noun	class
"3 years old"	Adjective	Attribute
"enters"	verb	Operation
"depends on...."	Intransitive verb	Operation (Event)
"is a" , "either..or", "kind of..."	Classifying verb	Inheritance
"Has a ", "consists of"	Possessive Verb	Aggregation
"must be", "less than..."	modal Verb	Constraint

Generation of a class diagram

Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than \$50. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

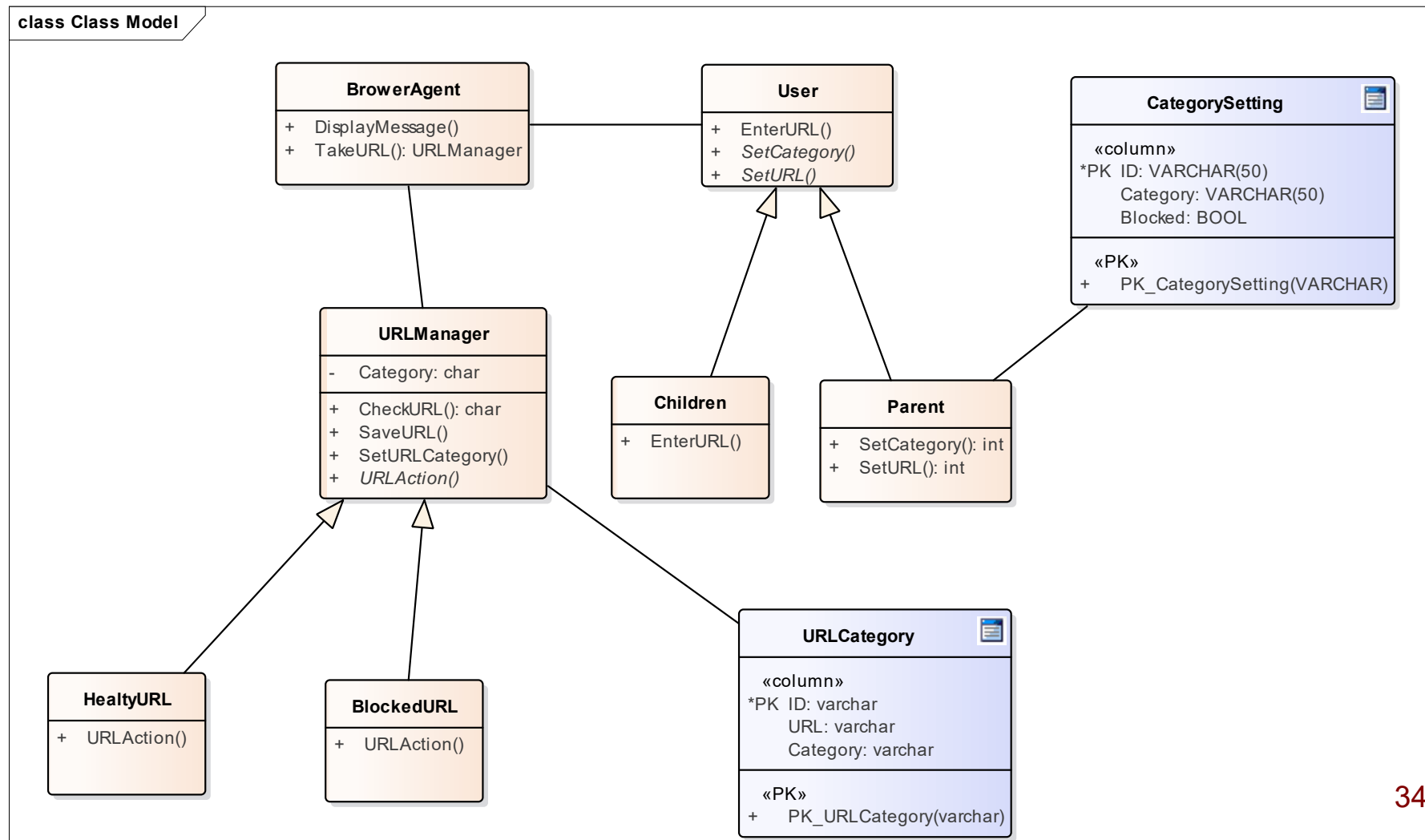
An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store



Exercise: Finding classes

- The E-Guard shall block unhealthy websites
 - Child opens Web browser Chrome
 - Child enters [www.#####.com](#) in the address box
 - E-guard takes the URL and checks the address DB for category
 - It shows this URL belongs to a blocked category set by parent
 - E-guard blocks the site and shows a warning message in browser

- Child opens Web browser Chrome
- Child enters www.#####.com in the address box
- E-guard takes the URL and checks the address DB for category
- It shows this URL belongs to a blocked category set by parent
- E-guard blocks the site and shows a warning message in browser





State Diagrams

Statechart Diagrams

- A Statechart is a notation for describing the sequence of states an object goes through in response to external events
- A State is a condition satisfied by the attributes of an object. It depicted by a rounded rectangle
 - An Incident object has four states: Active, Inactive, Closed and Archived.
- A transition represents a change of state triggered by events and is depicted by open arrows connecting two states
- A Small solid black circle indicates the initial state
- A circle surrounding a small solid black circle indicates a final state

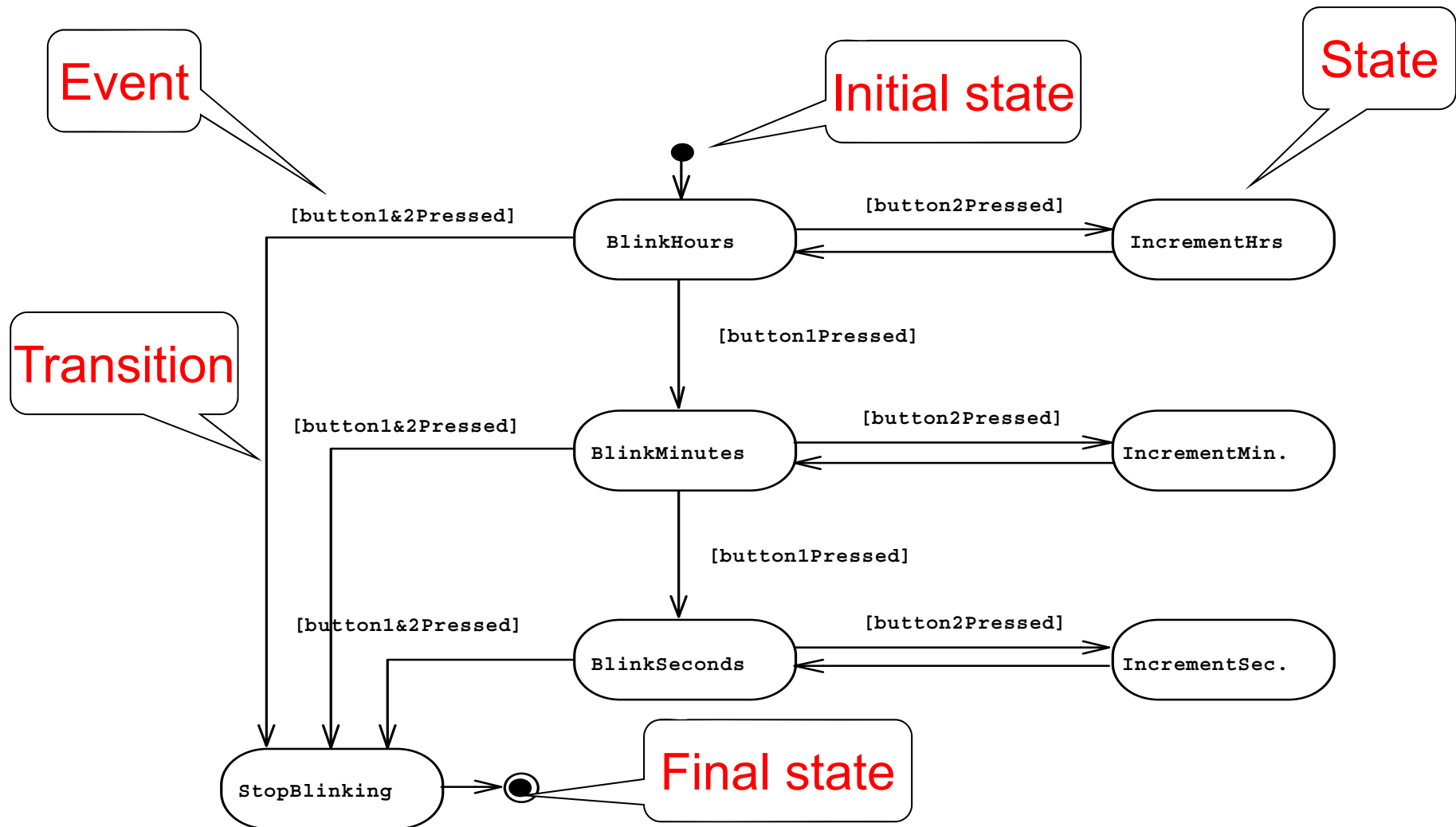
class 2ButtonWatch

2ButtonWatch

- Hours: int
- Minutes: int
- Seconds: int

- + button1&2Pressed()
- + button1Pressed()
- + button2Pressed()

State Chart Diagrams



Represent behavior as states and transitions

Statechart Diagrams Summary

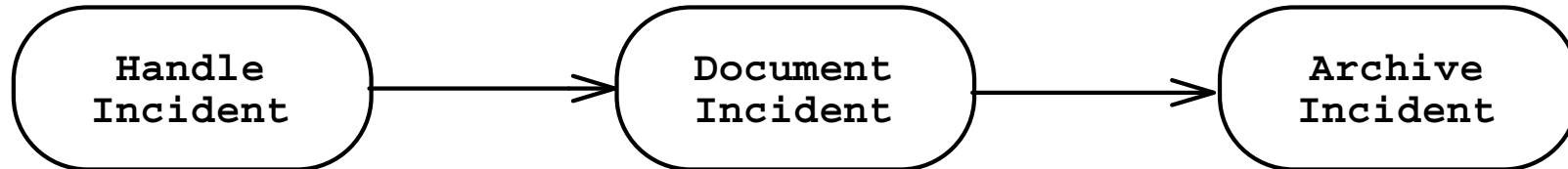
- Statechart diagrams are used to represent nontrivial behavior of a subsystem or an object
- What is the difference between interaction diagrams and Statechart Diagrams
 - Statechart diagrams make explicit which attribute or set of attributes have an impact on the behavior of a *Single* object
 - Interaction Diagrams are used to identify participating objects and services they provide.



Activity Diagrams

Activity Diagrams

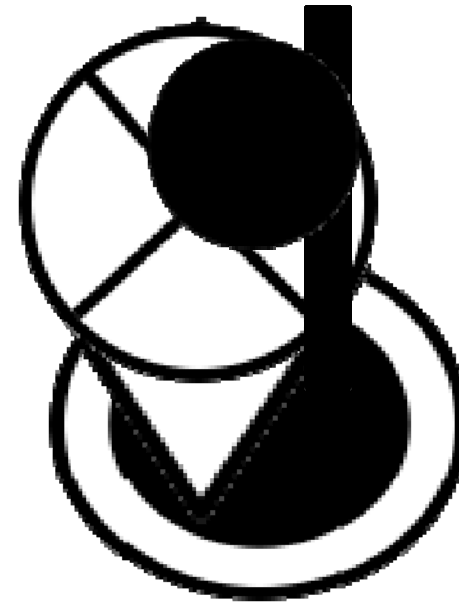
- An activity diagram shows flows of control among activities and actions associated with a particular object or set of objects



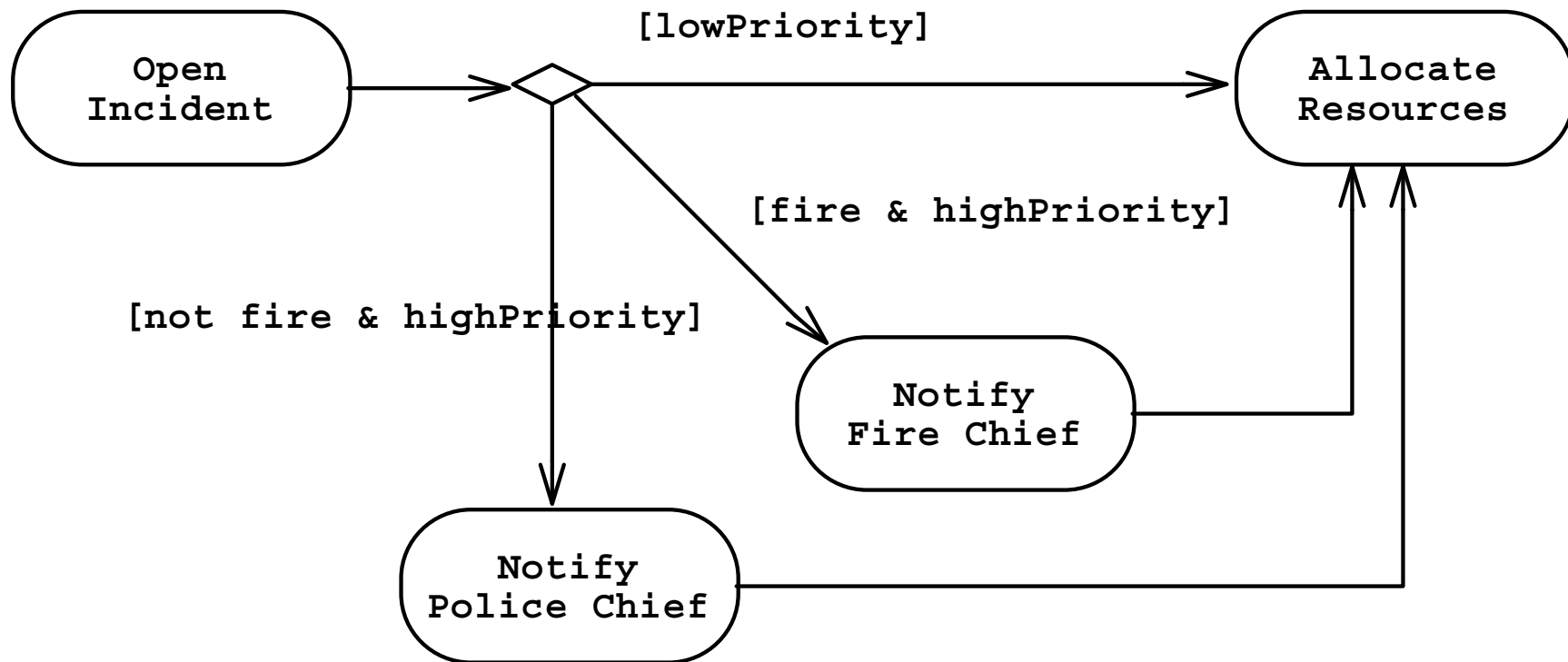
- Modelers typically use activity diagram to illustrate the following
 - The flow of complicated use case
 - A workflow across use cases
 - The logic of an algorithm

Control Nodes in an Activity Diagram

- Initial node
- Final node
 - Activity final node
 - Flow final node
- Fork node
- Join node
- Merge node
- Decision node



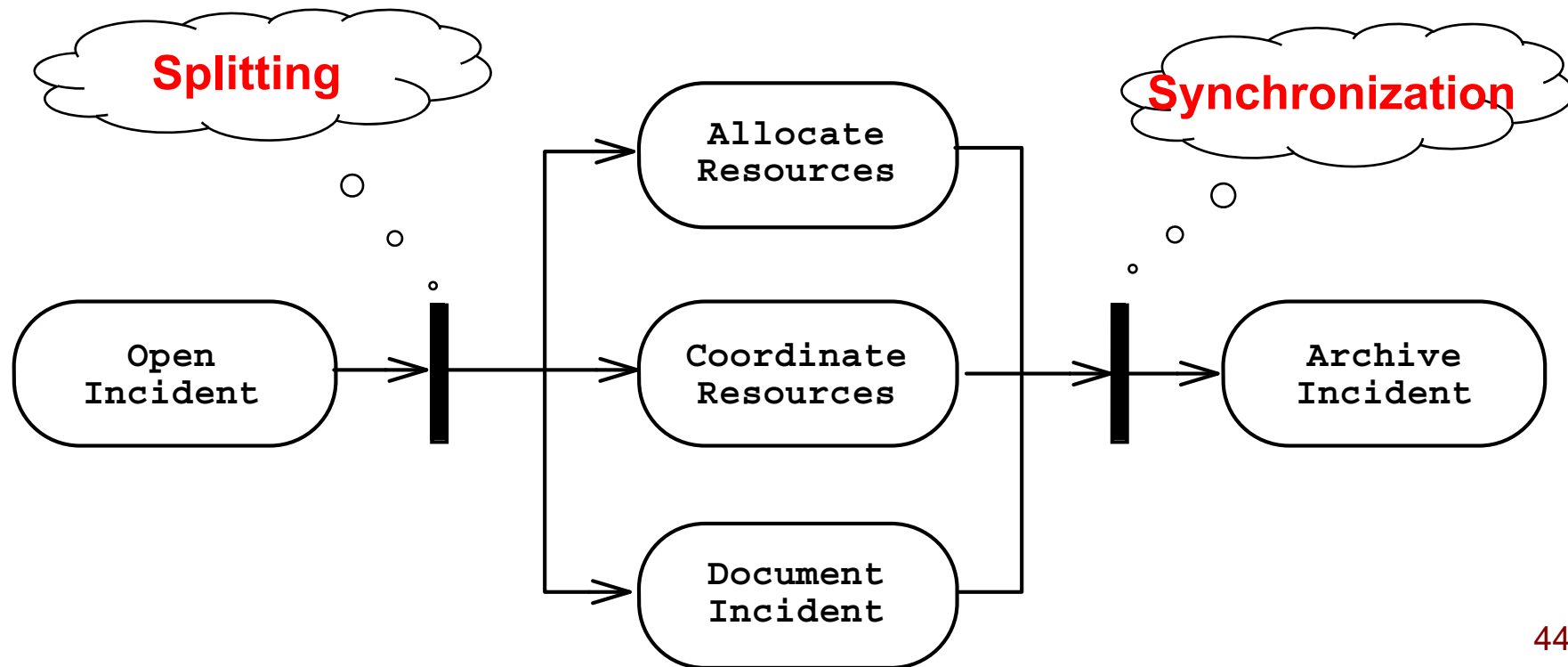
Activity Diagram: Modeling Decisions



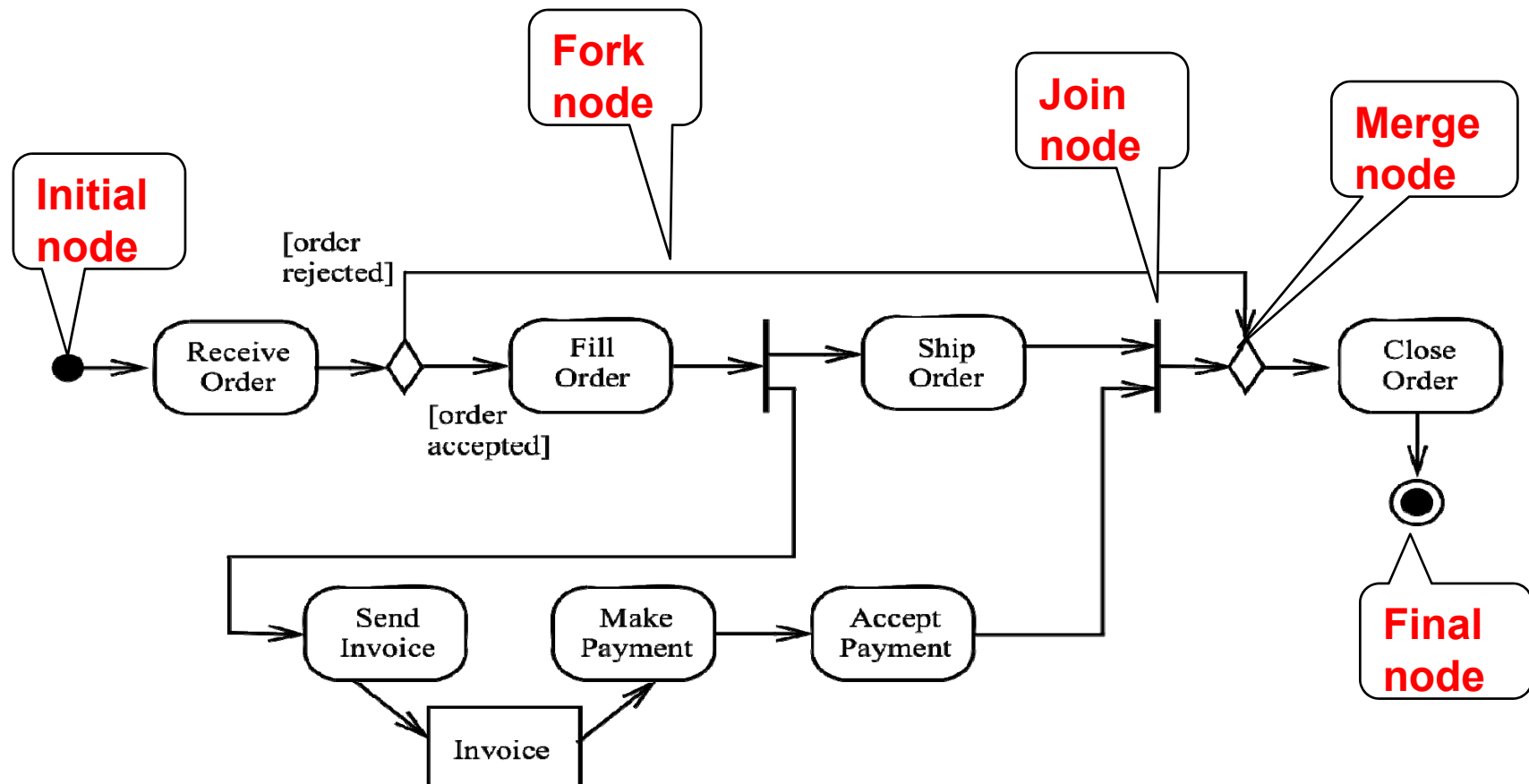
Decisions are branches in the control flow. They denote alternative transitions based on a condition on the state of an object or a set of objects. Depicted by a diamond with one or more incoming open arrows and two or more outgoing arrows

Activity Diagrams: Modeling Concurrency

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



Activity Diagram Example



Action Nodes and Object Nodes

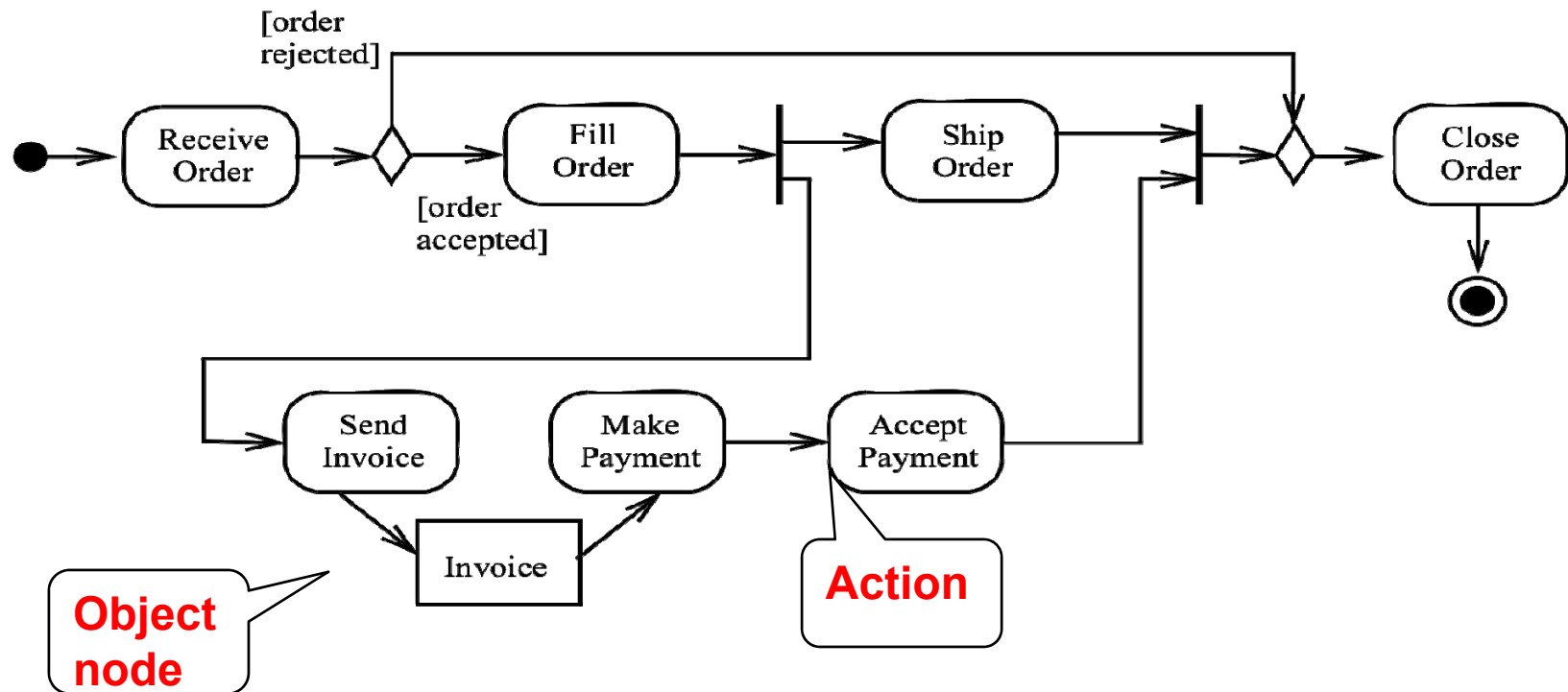
- Action Node

Action Name
- Object Node

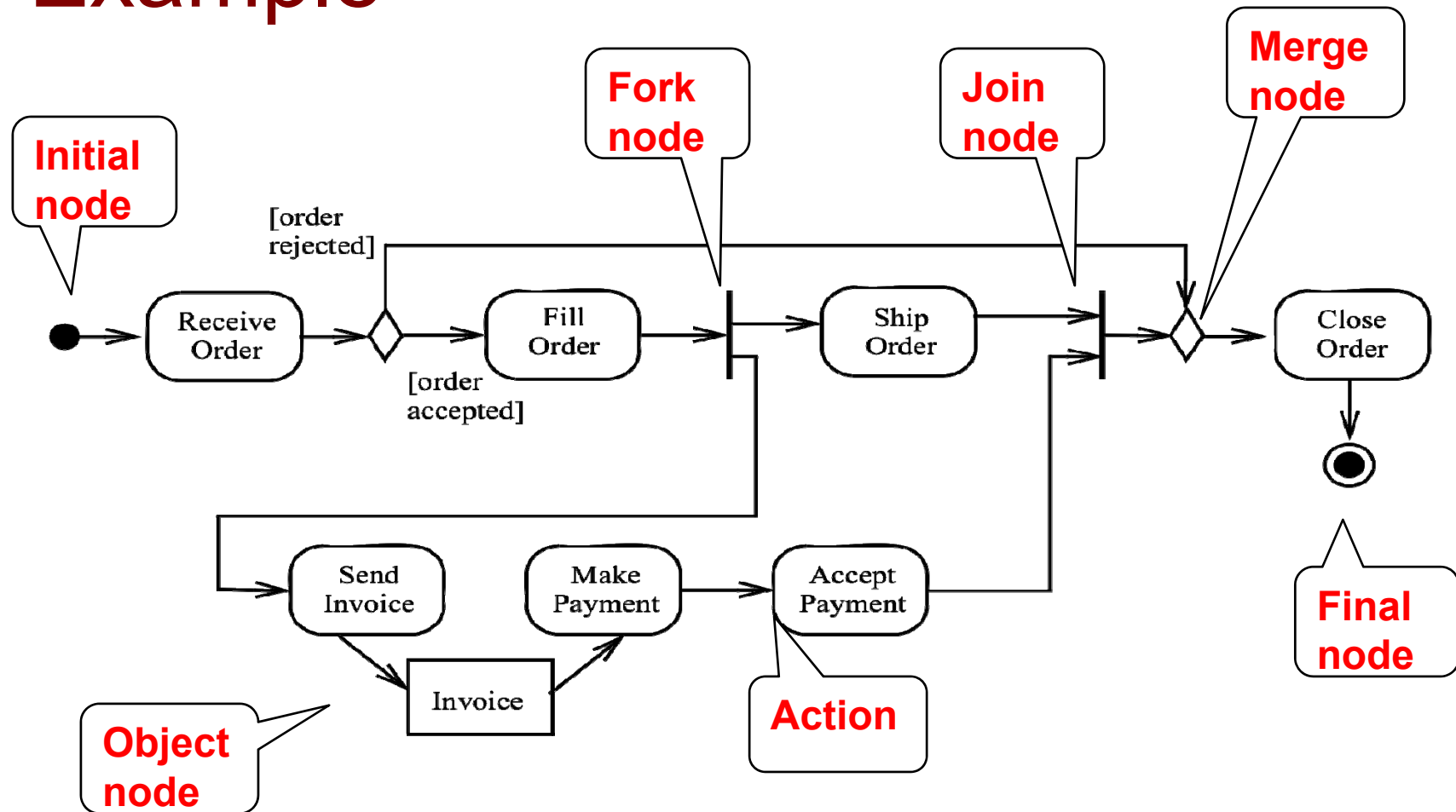
Object Name



Activity Diagram Example

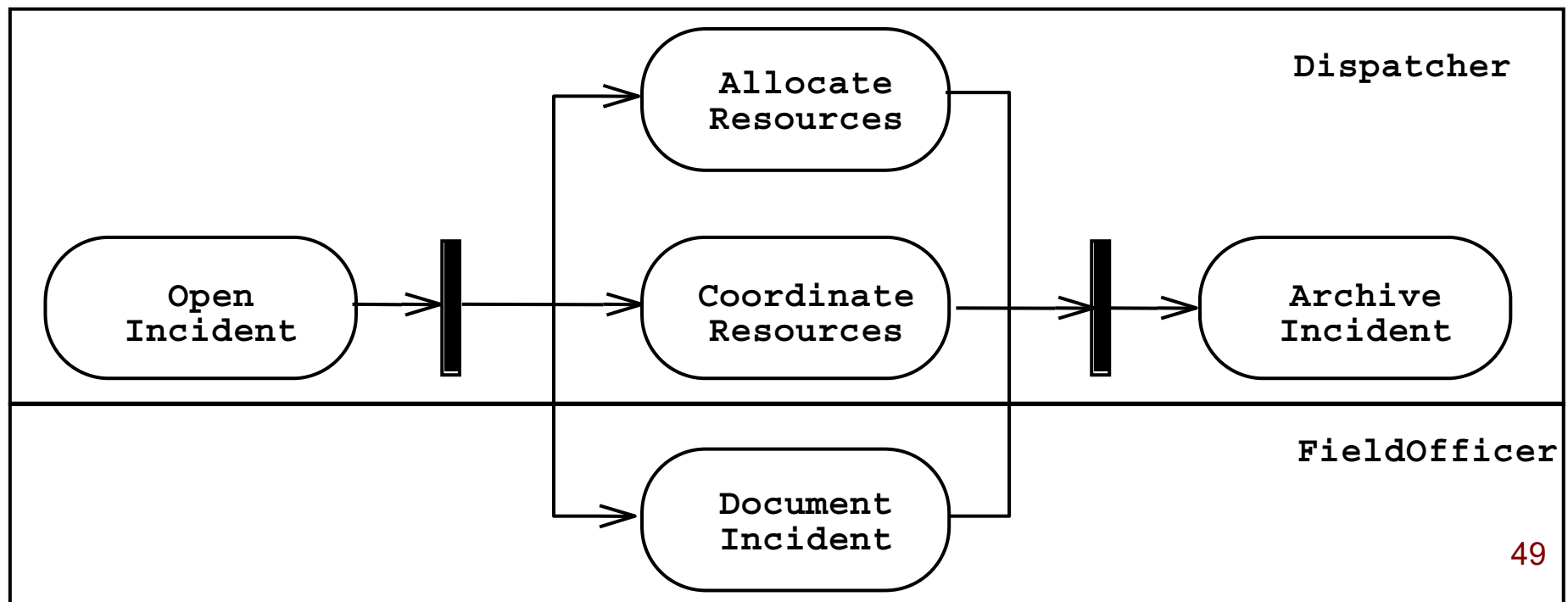


Summary: Activity Diagram Example



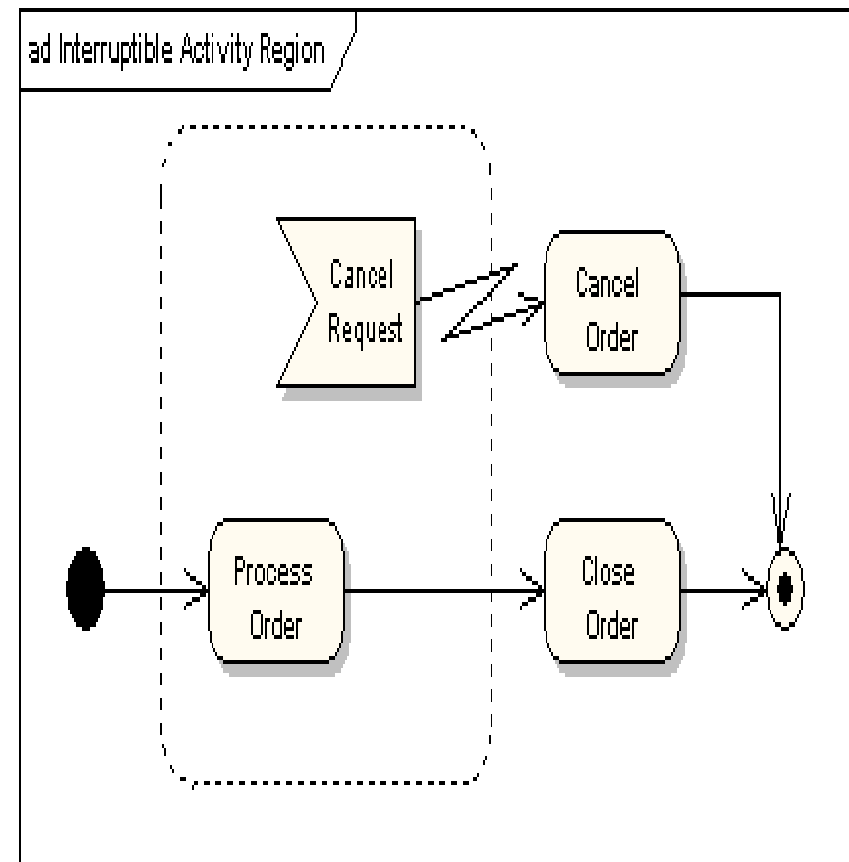
Activity Diagrams: Swimlanes

- Actions may be grouped into swimlanes to denote the object or subsystem that implements the actions.
- Dispatcher swimlane groups all actions that are performed by the Dispatcher object.



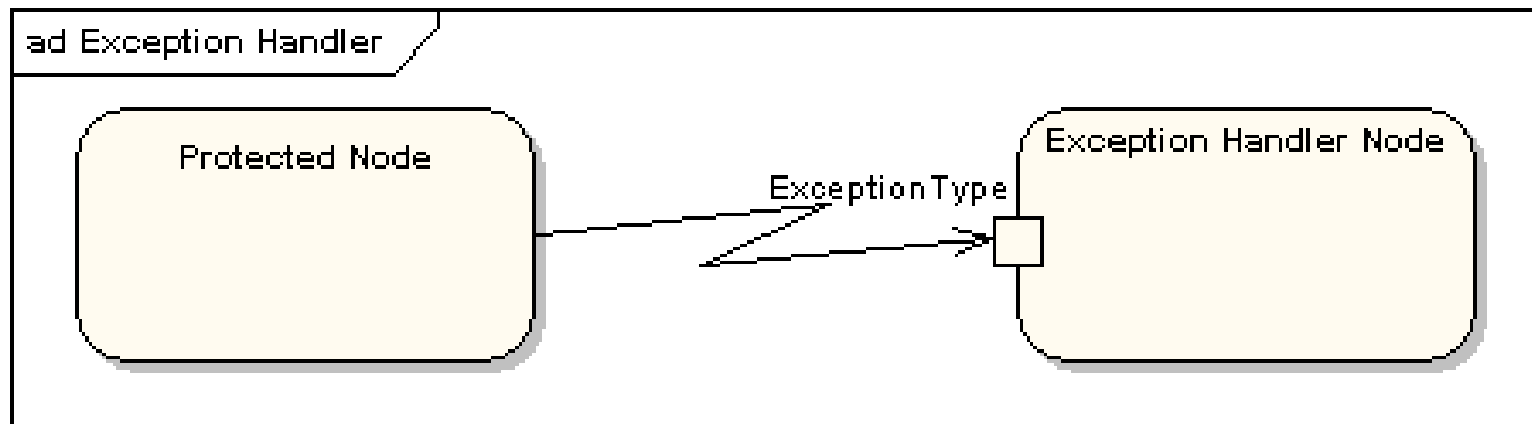
Activity Diagrams: Interruptible Activity Region

- an interruptible activity region surrounds a group of actions that can be interrupted
- example
 - "Process Order" action will execute until completion;
 - then it will pass control to the "Close Order" action, unless a "Cancel Request" interrupt is received, which will pass control to the "Cancel Order" action



Activity Diagrams: Exception Handlers

- exception handlers can be modeled on activity diagrams as in the example below



Activity Diagram Exercise

- *Consider the process of ordering a pizza over the phone. Draw an activity diagram representing each step of the process, from the moment you pick up the phone to the point where you start eating the pizza. Do not represent any exceptions. Include activities that others need to perform.*

