An Introduction to the Unified Modeling Language

Omar E. Pérez, PMP
operez@us.ibm.com

Agenda

- Why UML?
- The Unified Modeling Language
  - Use Case Diagrams
  - Activity Diagrams
  - Interaction Diagrams
  - Class Diagrams
    - Relationships
- IBM UML Solution Quick Demo

Caveat Central

- This session **will not** teach you a specific software design process
- The UML **is a language**, just like any other language
- UML **is not a software development process**
- It is best to keep it simple
  - Only use the diagrams that your project needs
  - Provide enough information to make a point

The Unified Modeling Language

How did we get here, and why do we care?

- UML is just a baby
  - The UML is a result of combining different modeling frameworks from Jacobson, Rumbaugh, and Booch
  - The first UML specification was ratified in 1997
  - The current UML version is 2.0, which was formalized in 2004
- The UML is a language of **pictures**, used to create software **models**
- Using pictures allows non-technical stakeholders to participate in the modeling process
  - A picture is worth a thousand lines of code
- Models are **faster**, **cheaper**, and **easier** to change than code

Terminology

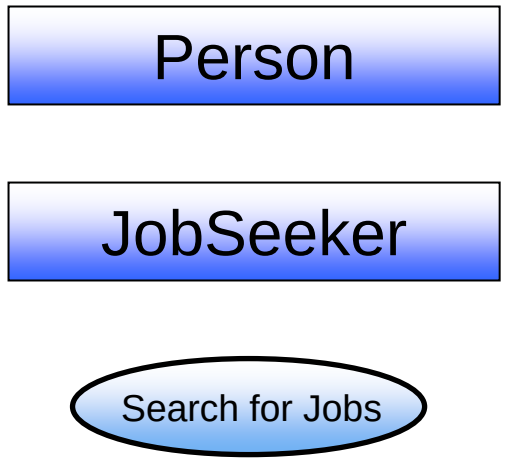
Object-Oriented Terminology

- Class
  - A pattern, template, or model for an object
  - A “cookie cutter”

- Object
  - An **instance** of a classifier
  - A combination of **data** and **functions**
  - A “cookie”
- Attributes
  - The **data** in an object (characteristics, fields)
- Operations
  - The **functions** in an object (procedures, subroutines, methods)

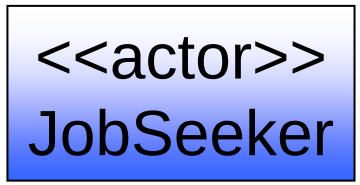
Classifiers

- A classifier is a category of UML elements that have some common features, such as attributes or operations
  - Can sometimes be used to refer to a class
- Types of UML classifiers
  - class
  - component
  - data type
  - interface
  - node
  - signal
  - subsystem
  - use case

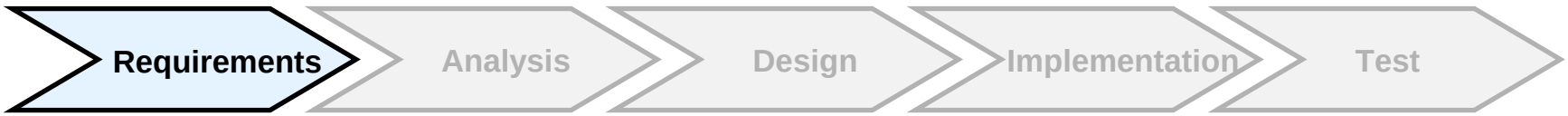


Stereotypes

- Provide an extensibility mechanism to UML
- Allows tagging existing modeling elements to create new ones
- Stereotypes must be defined
  - Within the model
  - By reference to external document
- You can stereotype both classes and relationships
- Each model element can have zero to many stereotypes
- Graphically rendered as a name enclosed between guillemets



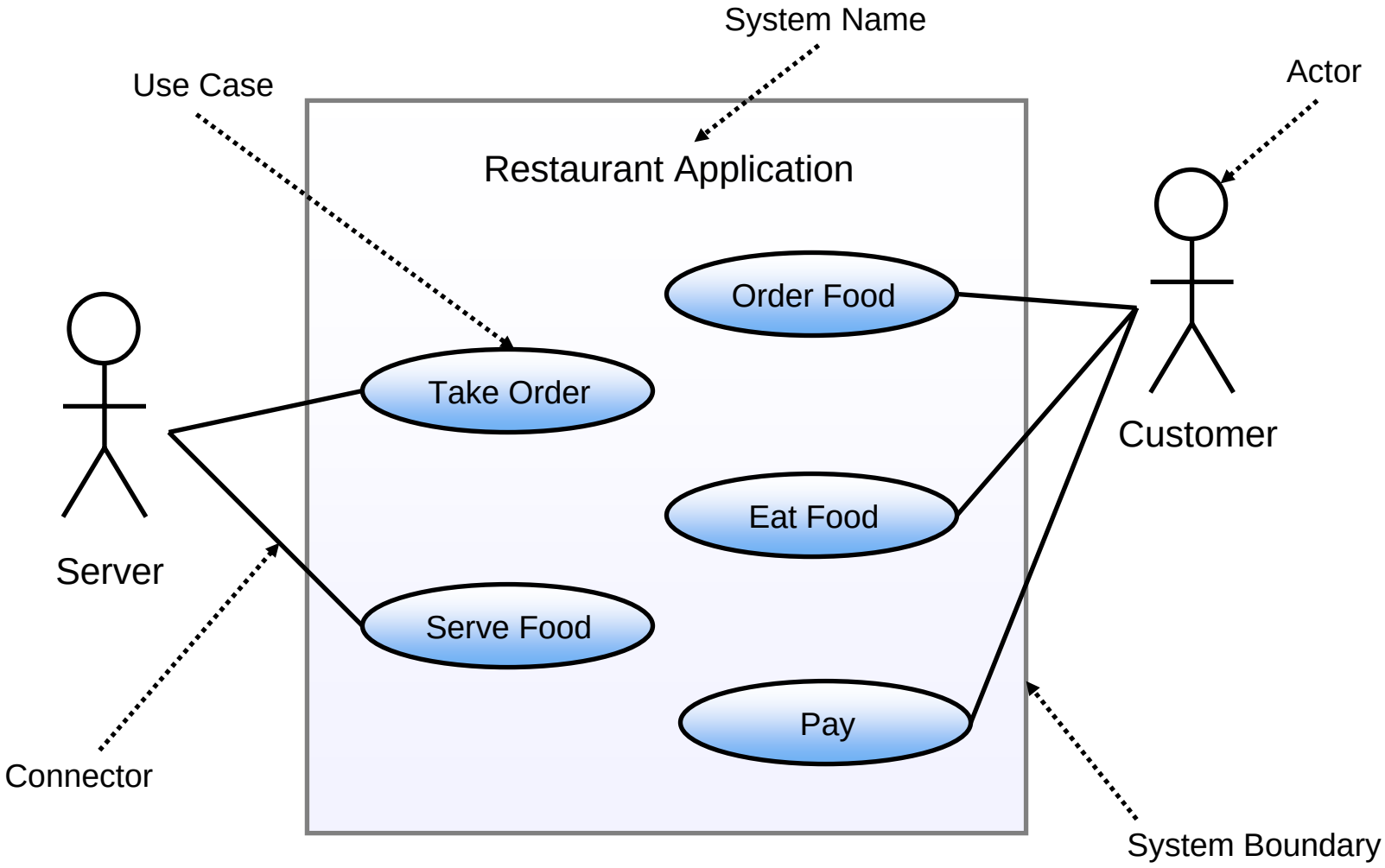
Use Case Diagrams



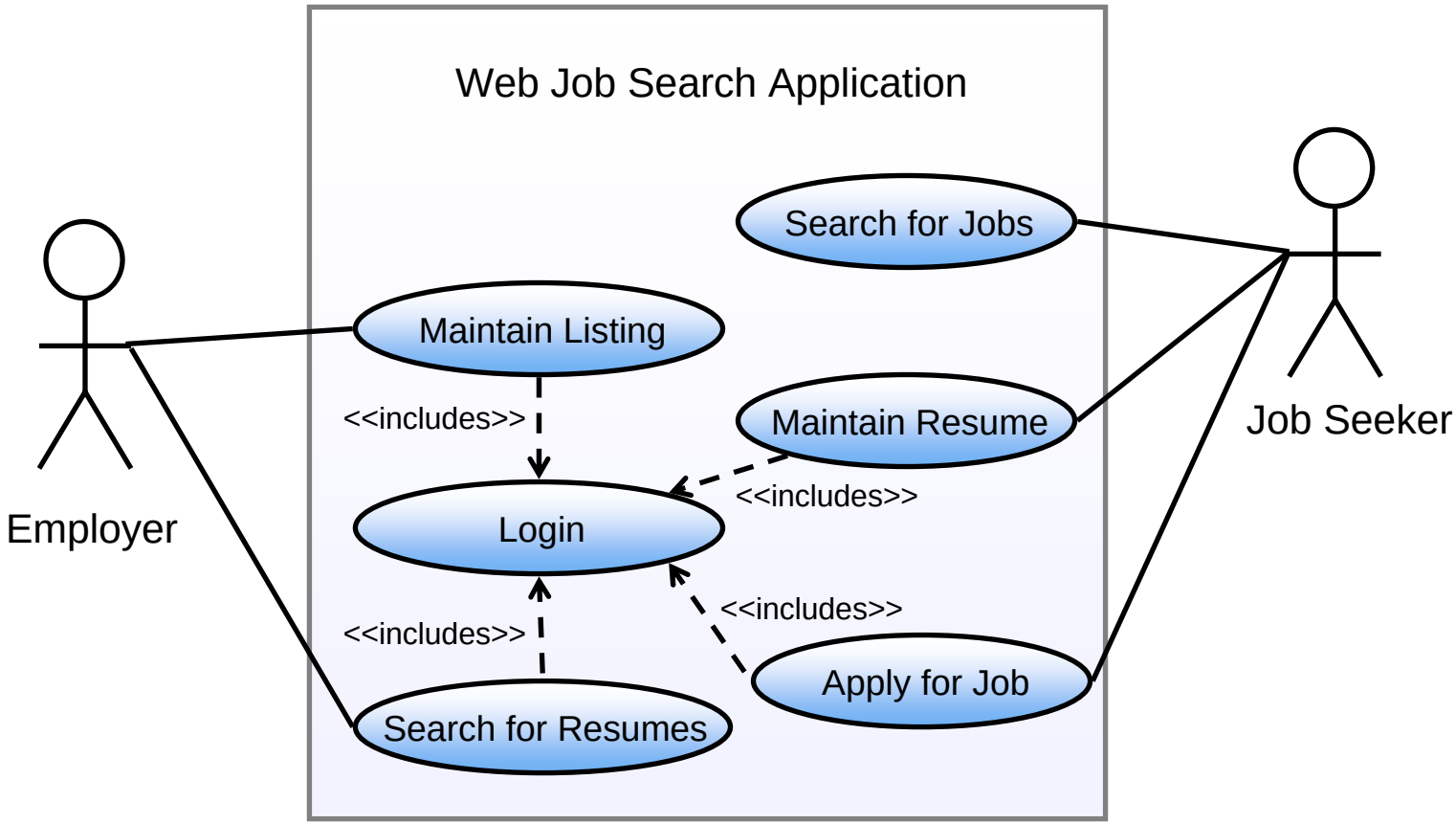
Use Case Diagrams: Gathering Requirements

- Illustrate relationships between use cases
- Use case diagrams consist of
  - Stick figures that represent **actors**
  - Ovals that represent the **use cases**
  - Connectors that indicate **relationships**
- The goals of use case diagrams are
  - Document important aspects of the system
  - Give a **big picture** view of the system
  - Facilitate requirement **prioritization**

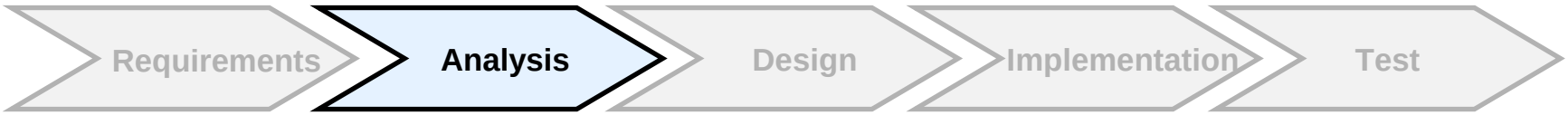
Use Case Diagram



Use Case Diagram: Example



Activity Diagrams



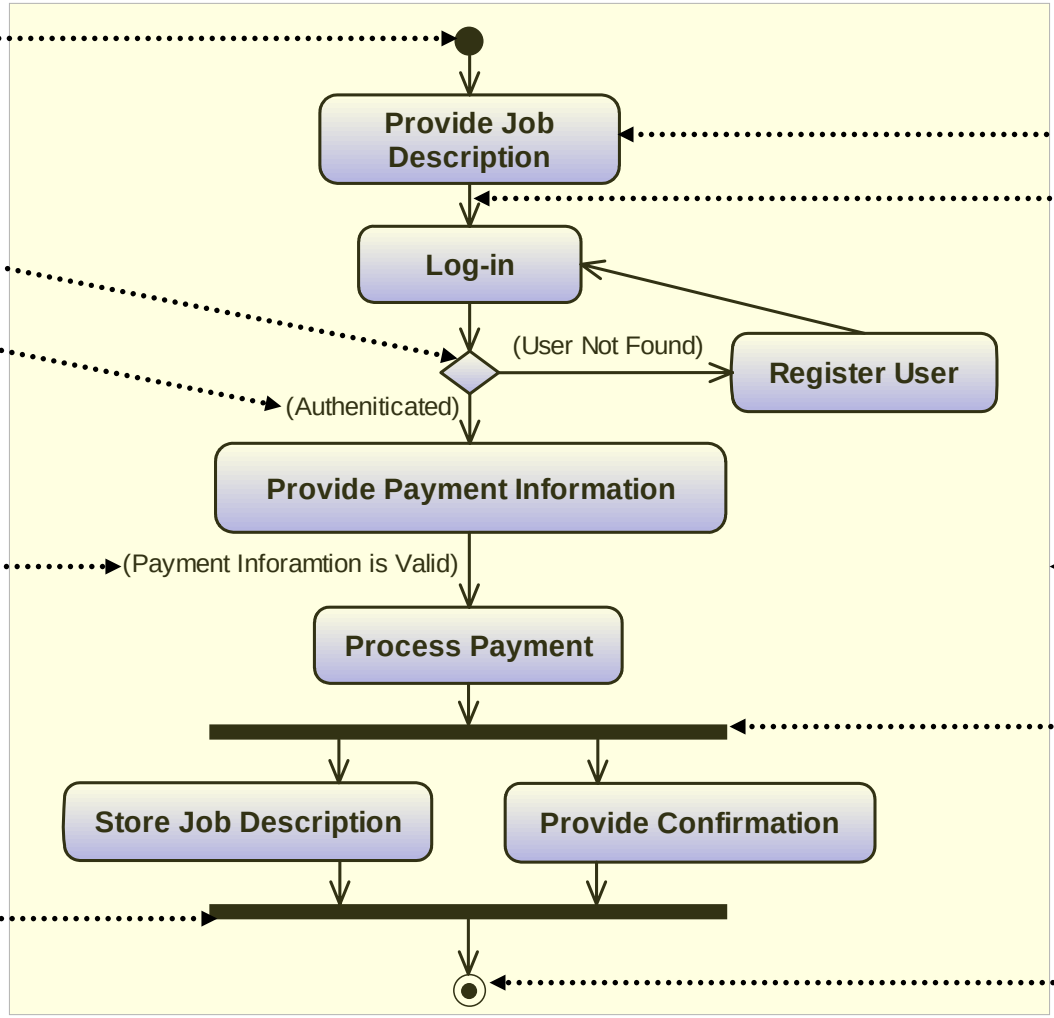
Activity Diagrams: “OO Flowcharts”

- Activity diagrams are flowcharts on steroids
- Used to **model processes**
- One common use is to **model a use case** as a series of actions
- Focus on communicating one **specific** aspect of the system
- Semantics based on **Petri Nets**
  - Flow of tokens around a network according to specific rules

Activity Diagrams: Nodes and Edges

- Activities are a collection of **nodes** connected by **edges**
- Three categories of nodes
  - Action node – discrete units of work that are atomic with the activity
  - Control node – control the flow through the activity (branching)
  - Object node – represent objects used in the activity (data)
- Two categories of edges
  - Control flow – represent the flow of control through the activity
  - Object flow – represent the flow of objects through the activity
- Tokens are moved from a source node to a target node across an edge

Activity Diagrams: Example



Initial node (control node)

Action node

Decision node (control node)

Edge (control flow)

Post condition

Guard condition

Activity

Fork (control node)

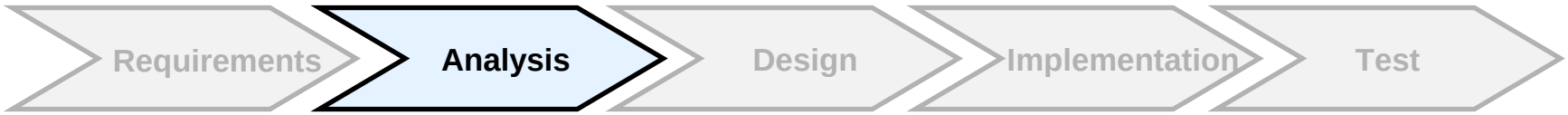
Join (control node)

Final node (control node)

State and flow in an activity diagram

- The state of the system at any point is determined by the position of its tokens
- Tokens represent
  - Flow of control
  - Objects
- Multiple initial nodes can exist
  - Flow starts at all initial nodes **simultaneously**
- First final node to be activated terminates all other flows and the activity itself

Interaction Diagrams



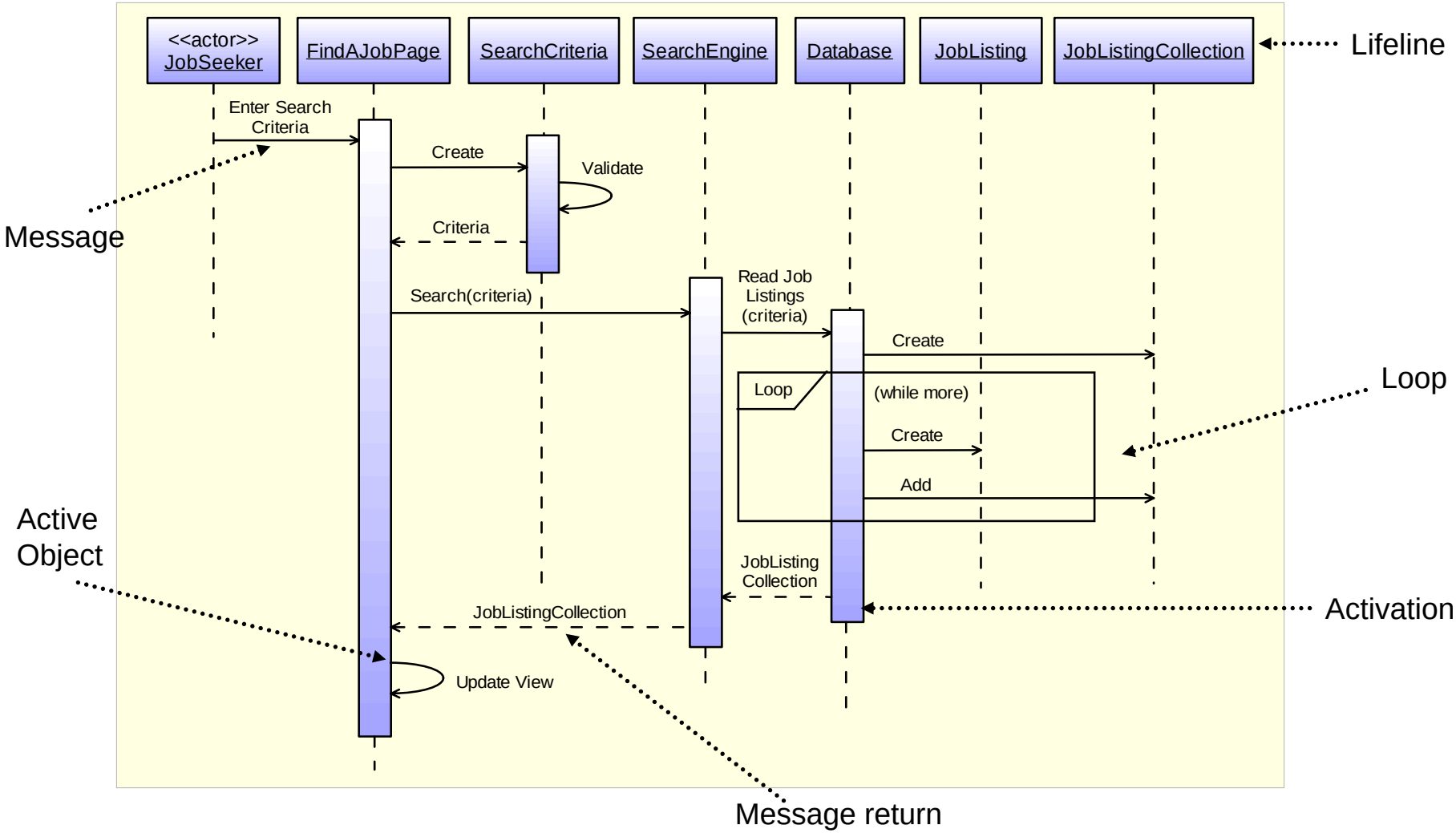
Interaction Diagrams: Discovering Behaviors

- Model interactions between parts of the system
- There are **four** types of interaction diagrams
  - **Sequence**
  - **Communication**
  - Interaction overview
  - Timing
- Demonstrate how a use case is realized by system components passing messages back and forth

Sequence Diagrams

- The **key elements** of sequence diagrams are
  - Lifelines – a participant in an interaction
  - Messages – a specific communication between lifelines
- Sequence diagrams show **interactions between lifelines** as a time-ordered sequence of events
  - Time flows from top to bottom
- In general, they are **easier to understand** than communication diagrams
- Tend to evolve with class diagrams and use cases
- Long thin rectangles along the tail of a lifeline indicate an **activation** which is when a lifeline has focus of control

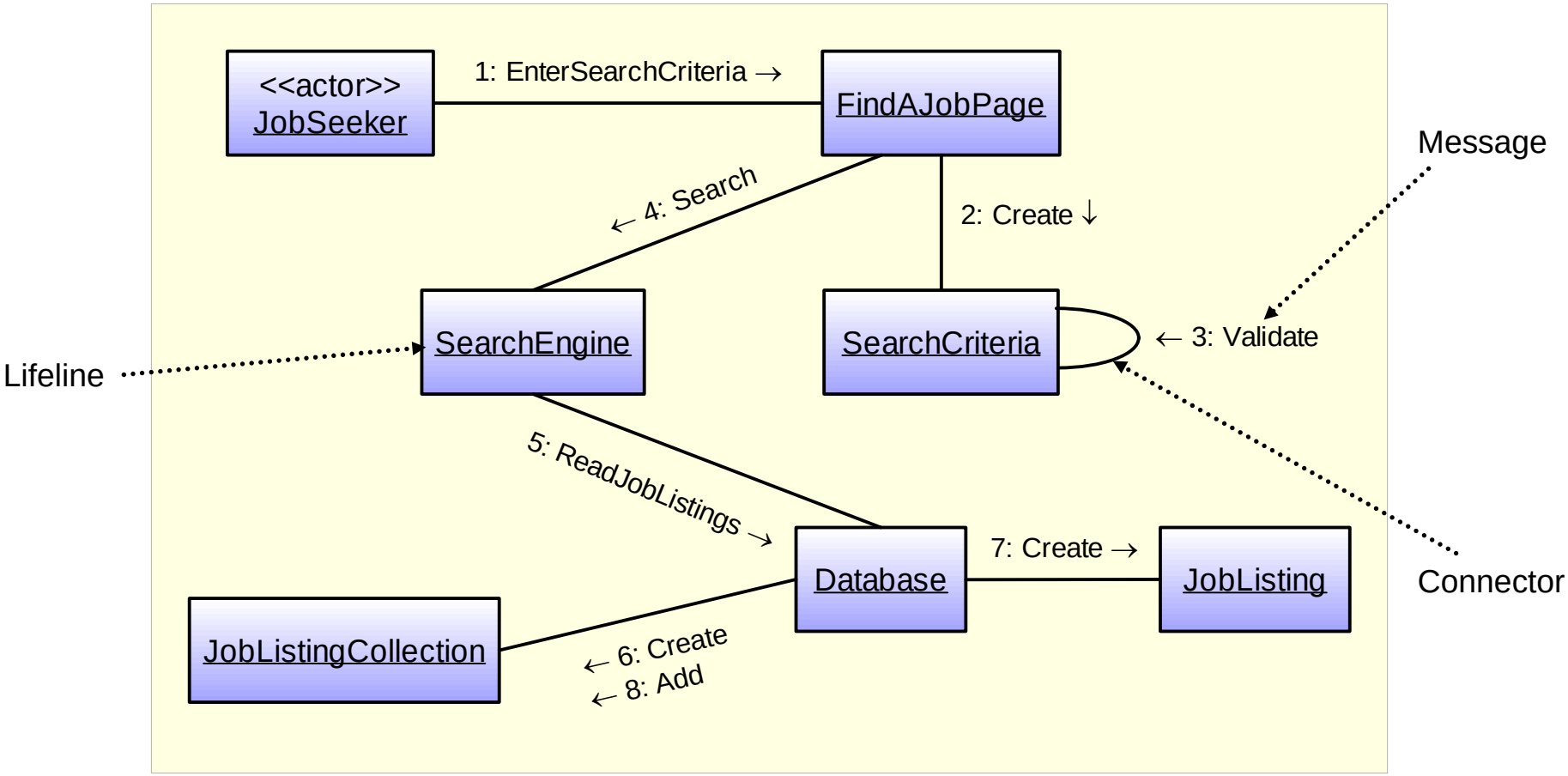
Sequence Diagrams: Example



Communication Diagrams

- Emphasize **structural aspects** of an interaction – how lifelines connect
- More compact and slightly **less detailed**
- Easier to manage when doing manual designs
- Subset of the functionality of sequence diagrams
- Lifelines are connected by links that provide communication channels for messages

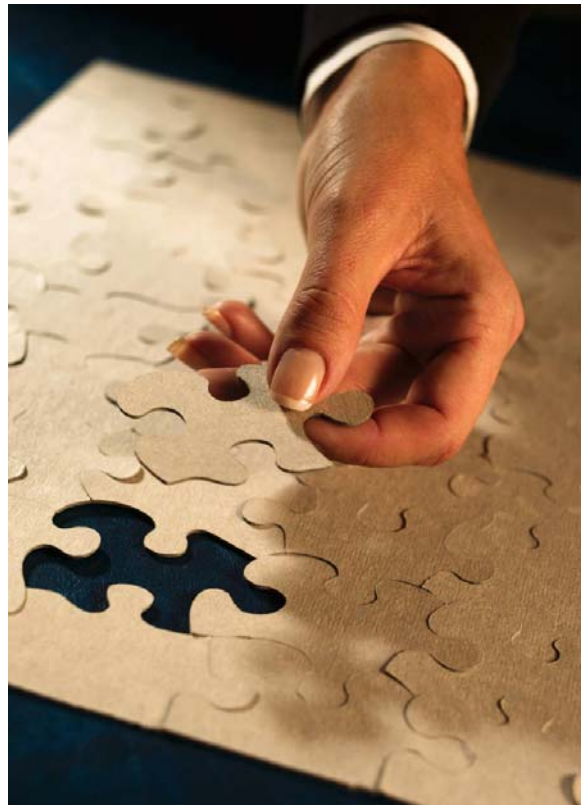
Communication Diagram: Example



Class Diagrams

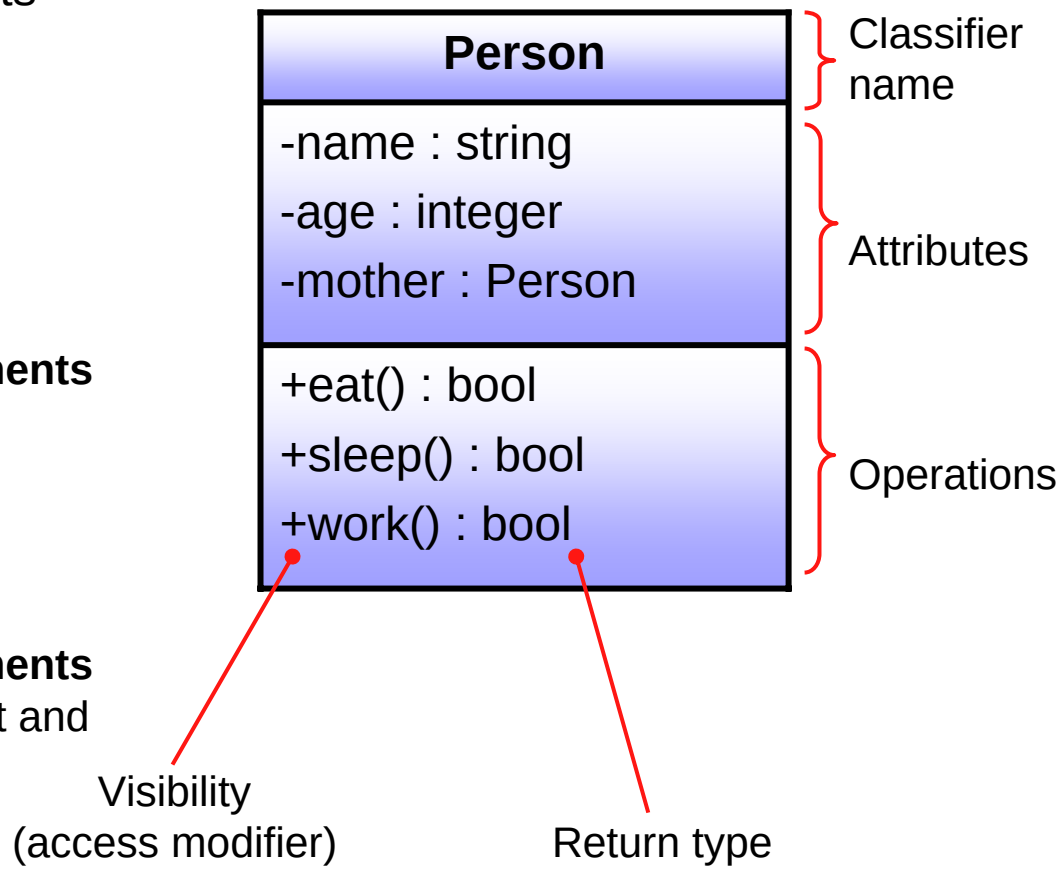
Class Diagrams

- **Most common UML diagram** and most important view of your design
- Class diagrams are **structure** diagrams
- Show the things that describe your problem and the **relationships** between those things
- Relationships are meaningful connections between modeling elements
- Types of relationships
  - Association
  - Aggregation
  - Composition
  - Generalization
  - Realization



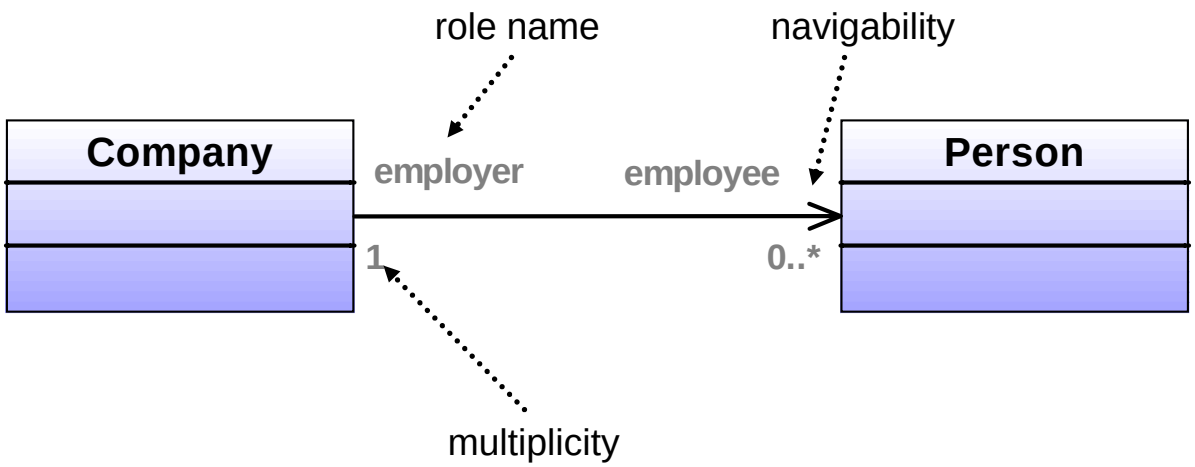
Simple Class Diagram

- Divided into three compartments
  - Classifier name
  - Attributes
  - Operations
- Attributes
  - Only requirement is the name
  - Can include additional **adornments** such as visibility, data type, multiplicity, and initial value
- Operations
  - Only requirement is the name
  - Can include additional **adornments** such as visibility, parameter list and return type



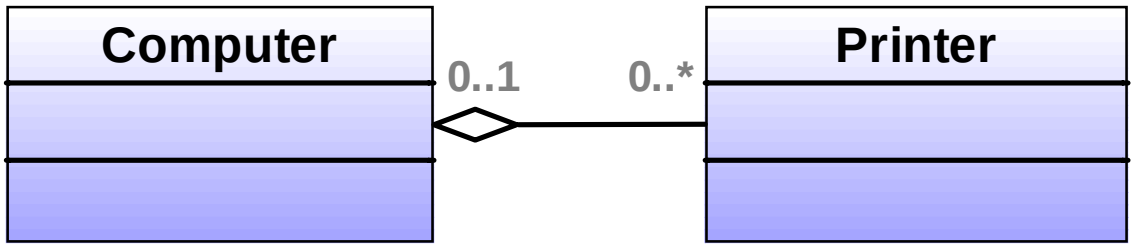
Association Relationship

- Basic relationship between classifiers
- Can have
  - name, role name, multiplicity, navigability (direction)



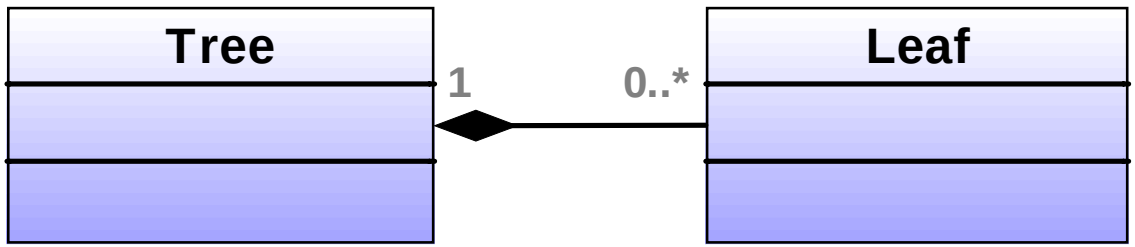
Aggregation Relationship

- Aggregation indicates a **loose type of relationship** between objects
  - a **whole-part** relationship
  - No life cycle dependency
- The syntax for indicating aggregation uses an **unfilled diamond** on the aggregate side of the connector
- A example might be a computer and its peripherals



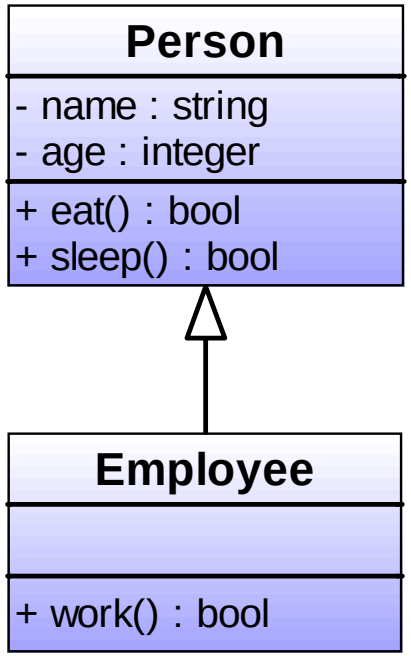
Composition Relationship

- Composition is a **stronger** form of aggregation
  - also a whole-part relationship
  - Has life cycle dependency
- The key difference is that the parts have *no* independent life outside of the whole
  - Each part belongs to one, and only one, whole
- The syntax for indicating a composition relationship uses a **filled diamond** on the composite side of the connector



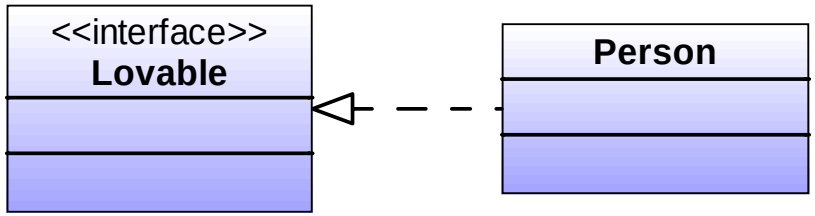
Generalization Relationship

- Generalization relationships are referred to using several terms
  - Parent-child relationship
  - “is-a” relationship
  - Inheritance
- In this relationship
  - The child can be substituted for the parent
  - The child gets all of the features of the parent and then can add features of its own
- Indicated in the UML by an open triangle connector pointing to the parent class

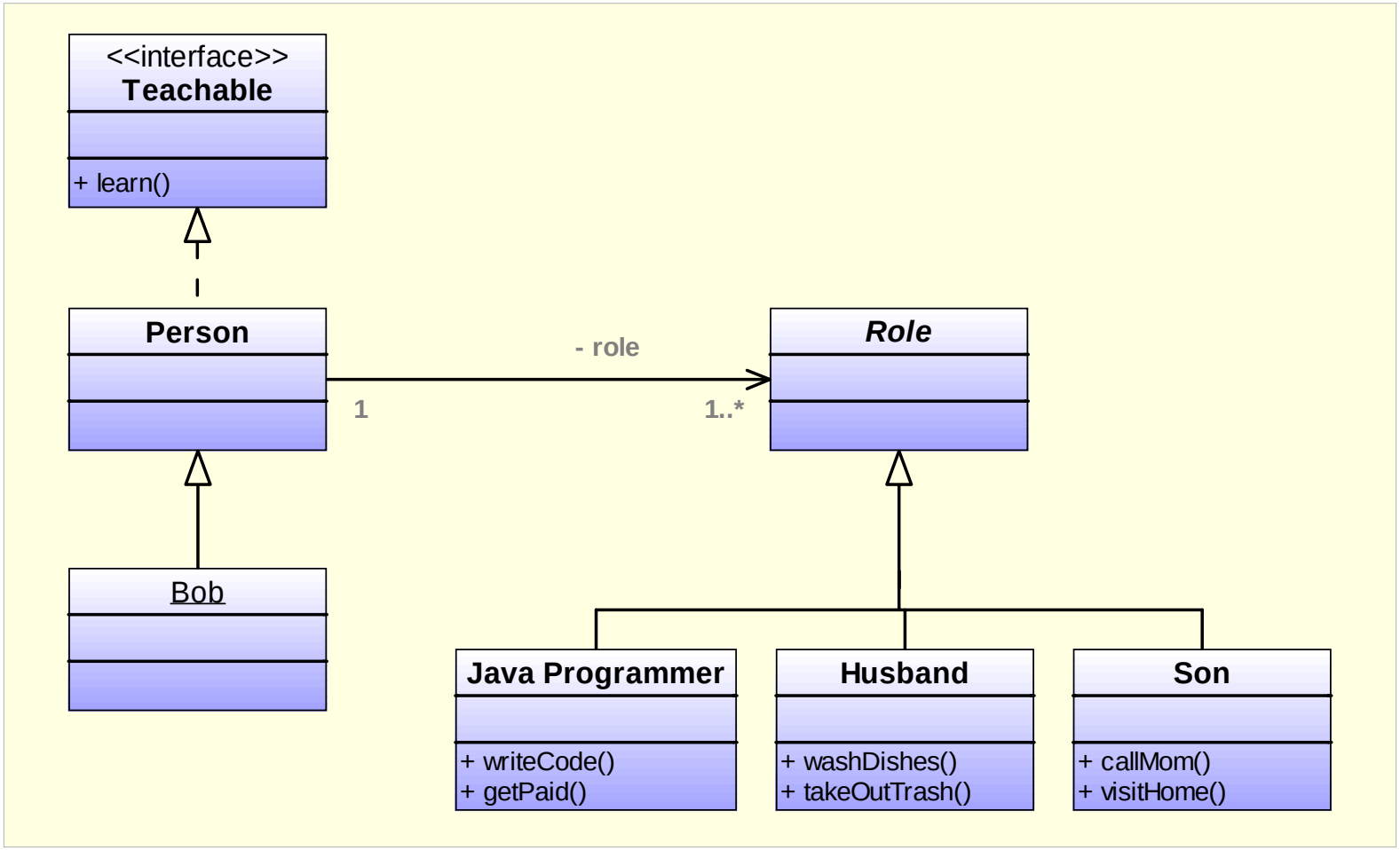


Realization Relationship

- Realization relationships refer to inheriting from or realizing **interfaces**
- A **contract** in which the class **realizing** the interface agrees to provide an implementation of the features declared by the interface



Class Diagrams: Example

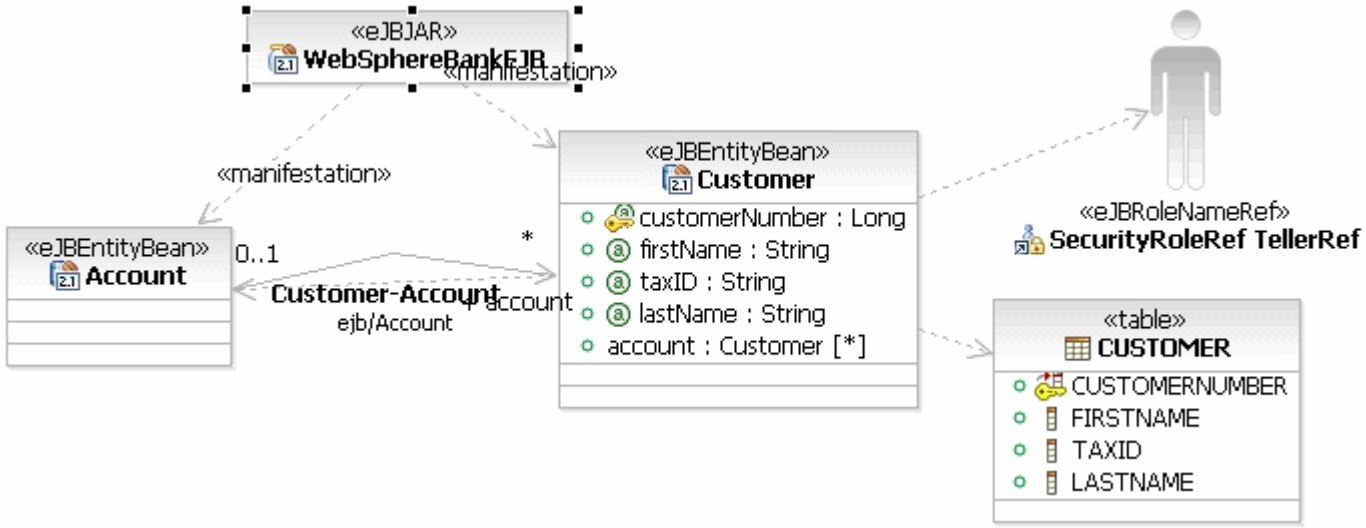


Finding the Classes You Need

- There are three kinds of classes
  - **Entity classes** represent data and things, are the easiest to find, and often pop out from the noun-verb analysis
  - **Control classes** generally act on other classes and facilitate communication between entity and boundary classes
  - **Boundary classes** connect elements inside the system to those outside the system
- Read through your use cases and identify the **nouns** and the **verbs**
  - Nouns become classes, verbs become operations
- Model use cases into activities to determine additional classes

IBM Solutions

- Use WebSphere Development Studio Client Advanced Edition to do **roundtripping**
  - Go from your UML diagrams to source code, and back!



Show Me

Tips to Help you Navigate the UML

- Avoid analysis paralysis
- Create only the diagrams you need
- Don’t be afraid to downsize
- Use text sparingly, except when describing use cases
- Get a second opinion
- Perfection is overrated

Summary

Summary

- The Unified Modeling Language is a **standardized** set of **abstractions** for **object-oriented** software design
  - Can be adapted to model business and other processes
- The UML consists of diagrams. We covered:
  - Use Case Diagrams (behavior)
  - Activity Diagrams (behavior)
  - Interaction Diagrams
  - Class Diagrams (structure)

Appendix

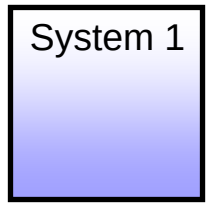
References

- IBM developerWorks
  - http://www.ibm.com/developerworks
- *UML Demystified* by Paul Kimmel
- The UML Resource Page
  - http://www.uml.org

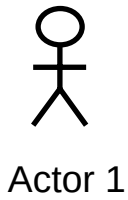
Use Case Diagrams: Syntax

- Ovals represent a use case

- A square represents the system boundary



- A stick figure represents an actor



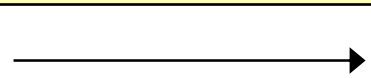
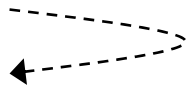
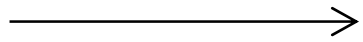
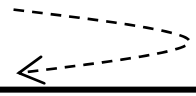
- Lines associate actors with their use case

Activity Diagrams: Syntax

- The **initial node** is a small filled circle with an edge coming out of it
- **Control flow** is represented by arrows called edges or flows
- A rectangle with rounded corners is an **action node**
- **Connector nodes** are labeled circles that have only either incoming or outgoing flows, not both
- Nodes can have **guard conditions,** indicated by parenthetical phrases on edges
- **Preconditions** and **postconditions** are represented as UML notes
- **Decision** and **merge** nodes are represented by diamonds
- **Forks** and **joins** are horizontal bars that represent parallel behavior
- Rectangles used to show who or what is responsible for an activity are called **activity partitions** or **swimlanes**

Sequence Diagrams: Syntax

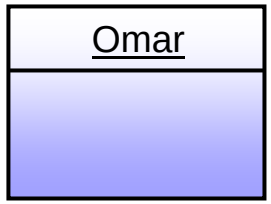
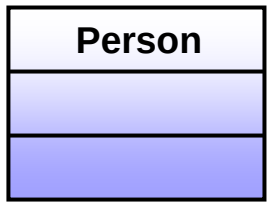
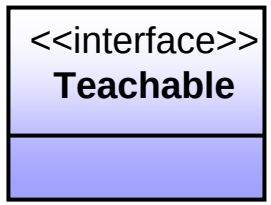
- A **lifeline** is a rectangle with a vertical line descending from it
  - The lifeline can represent an instance of a class or an actor in a use case
- **Messages** are directed lines connecting lifelines

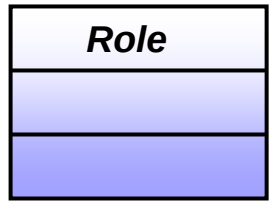
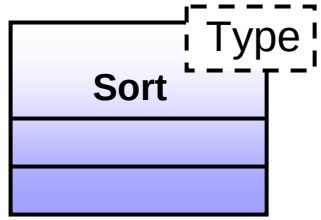
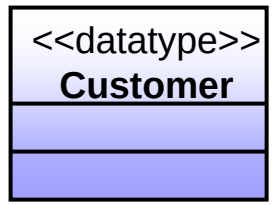
Message Type	Symbol
Synchronous Message	
Nested Message	
Asynchronous Message	
Return Message	
Nested Return Message	

Communication Diagrams: Syntax

- Communication diagrams consist of rectangular **classifiers** that represent objects
- Related classifiers are attached with lines called **connectors**
- Connectors are labeled with **messages** that include:
  - An arrow to indicate the direction of flow
  - A number to indicate the position in sequence

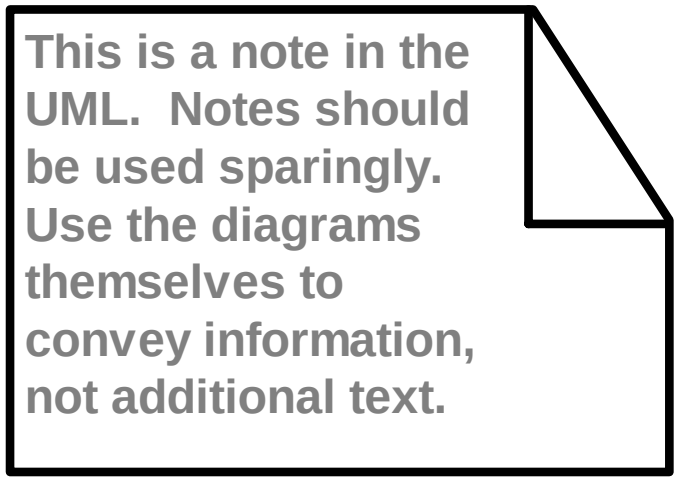
Class Diagrams: Syntax

Classifier	Symbol
Ojbect – an instance of a class	
Class	
Interface	

Classifier	Symbol
Abstract Class	
Generic or Template	
Persisted Entity	

Using Notes in UML Diagrams

- A note in the UML is indicated by a dog-eared rectangle



Trademarks and Disclaimers

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AS/400	**e-business on demand**	**i5/OS**
AS/400e	**IBM**	**OS/400**
eServer	**IBM (logo)**	**System i5**
	iSeries	

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, other countries, or both.
Intel, Intel Logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.