UML

Introduction

- UML = Unified Modeling Language
- It is a standardized visual modeling language.
  - Primarily intended for modeling software systems.
  - Also used for business modeling.
- UML evolved from earlier competing modeling languages.
  - Based on the best parts of those earlier methods.
  - Has continued to evolve since its creation.
- UML is **NOT** a visual programming language.

Architectural Views of UML

(part 1 of 3) User and structural views

- UML is centered around a number of different types of diagrams, each modeling the system from a different perspective.
  - *Use case diagrams* model the functionality of the system from the users' perspective.
  - Structural diagrams model the static structure of a system.
    - *Class diagrams* show the overall structure.
    - *Object diagrams* show the structure at a particular time.

Architectural Views of UML

(part 2 of 3) Behavioral view

- Interaction diagrams model the interaction of objects as they perform some operation.
  - *Sequence diagrams* model the sequences of messages that are sent between objects to carry out some operation.
  - *Collaboration diagrams* show the roles objects play in carrying out some operation.
- Behavoiral diagrams model the behavoir of objects.
  - A *state diagram* models the states an object can be in and the stimuli that cause it to change states.
  - *Activity diagrams* show how the behaviors of objects involved in some operation depend on each other.

Architectural Views of UML

(part 3 of 3) Implementation and environment views

- Physical diagrams show how the parts of a system are organized in the real world.
  - A *component diagram* shows the organization of the parts of the system into packages.
  - *Deployment diagrams* display the physical locations of the components of the system.

Why Use UML?

- Communicate information about a system.
  - Diagrams can be understood by non-programmers.
  - Models can serve as a blueprint for a system.
  - Models can help document a system.
- Even if the diagram itself is ultimately discarded, the act of creating it is useful since it helps you to understand whatever it is you're modeling.

Use Case Diagrams

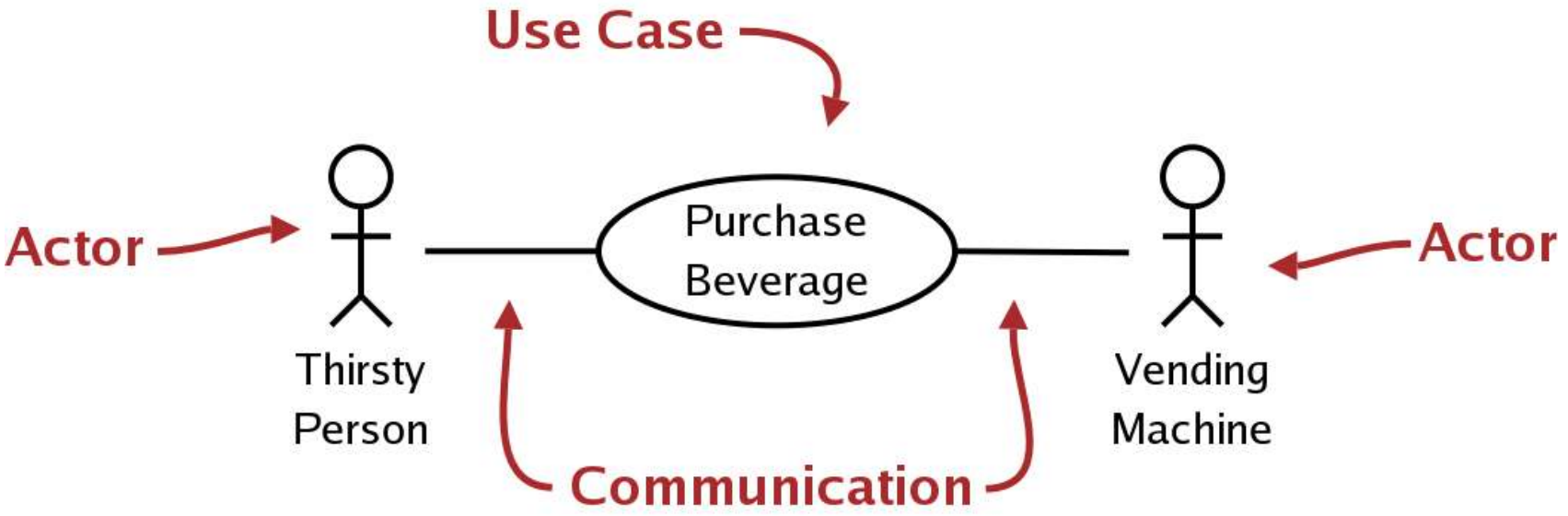
(part 1 of 5) What are they and what are tye used for

- A use case diagram models the users' view of the system.
  - Describes what the system does, not how it does it.
  - Shows how the user interacts with the system.
- Useful for:
  - Determining features.
  - Communicating with clients.
  - Generating testcases.

Use Case Diagrams

(part 2 of 5) Basic parts

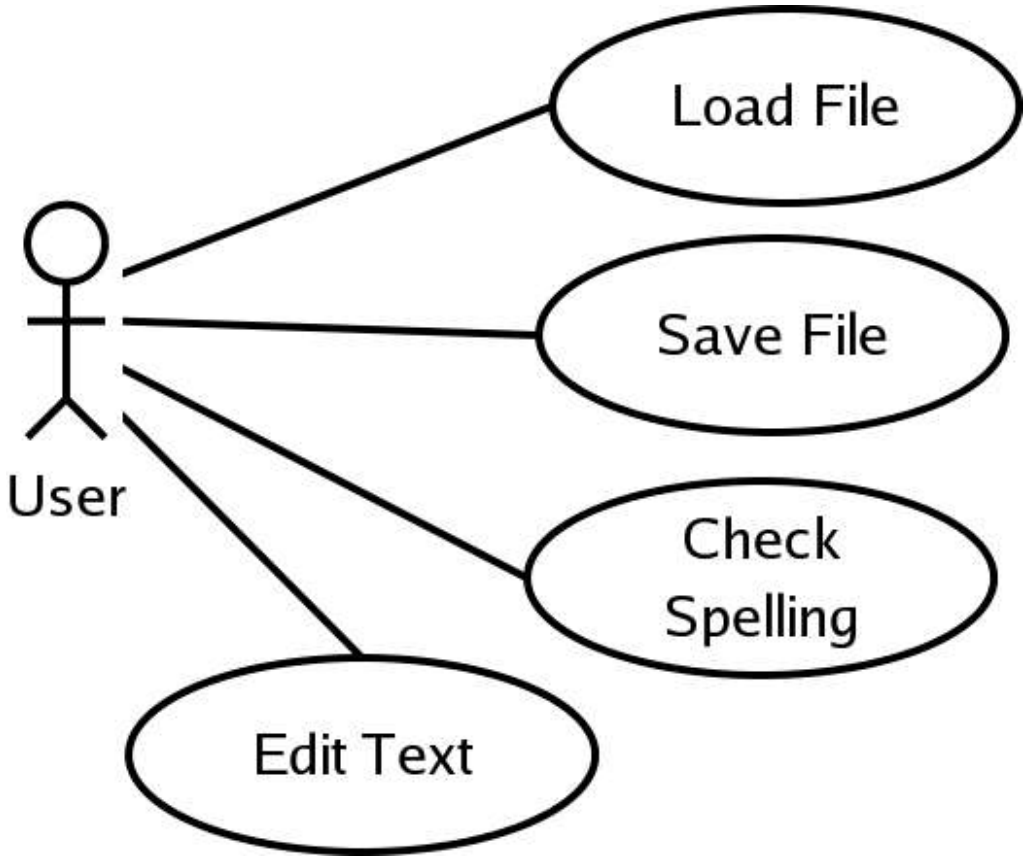
- Basic Vocabulary
  - Actor: A person or thing involved in some task
  - Use case: Something the user does with the system.
  - Communication: Lines linking actors and use cases.



Use Case Diagrams

(part 3 of 5) Simple Example

- Use case diagram for a text editor:



Use Case Diagrams

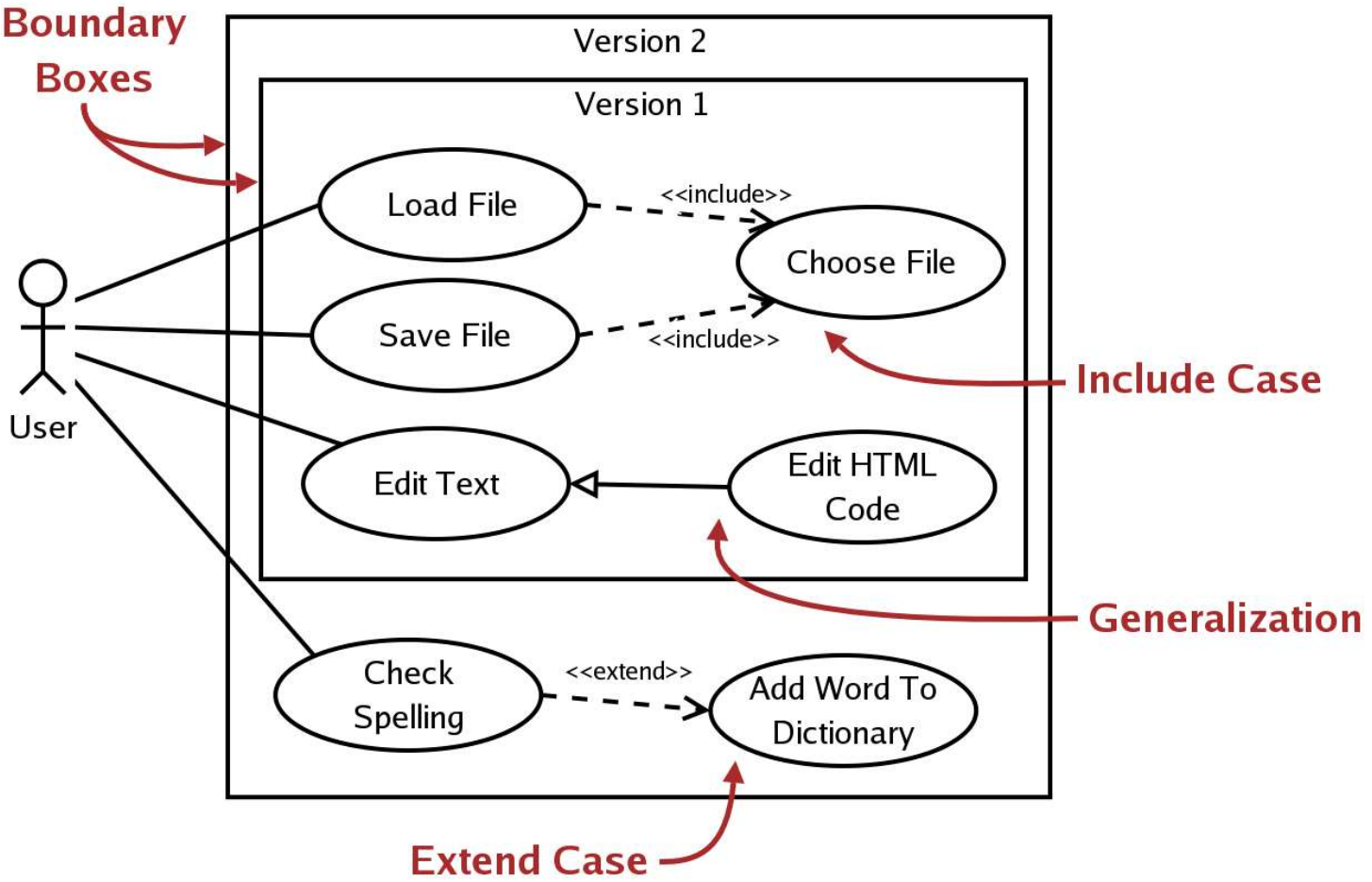
(part 4 of 5) More parts

- More vocabulary:
  - Include - Like a procedure call.
  - Extend - Like a procedure that is called sometimes depending on some condition.
  - Generalizations - A specialization of some case.
  - Boundary box - Group use cases together.
- Examples on next slide...

Use Case Diagrams

(part 5 of 5) More complex example

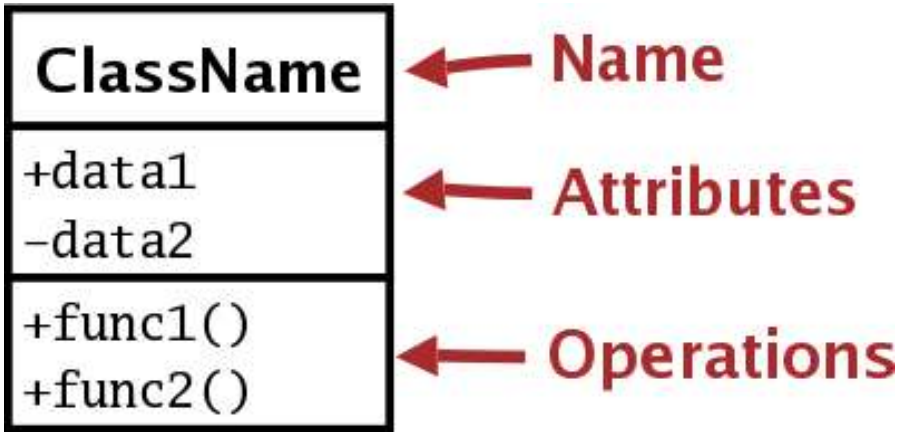
- Another use case diagram for a text editor:



Class Diagrams

(part 1 of 7) Class compartments

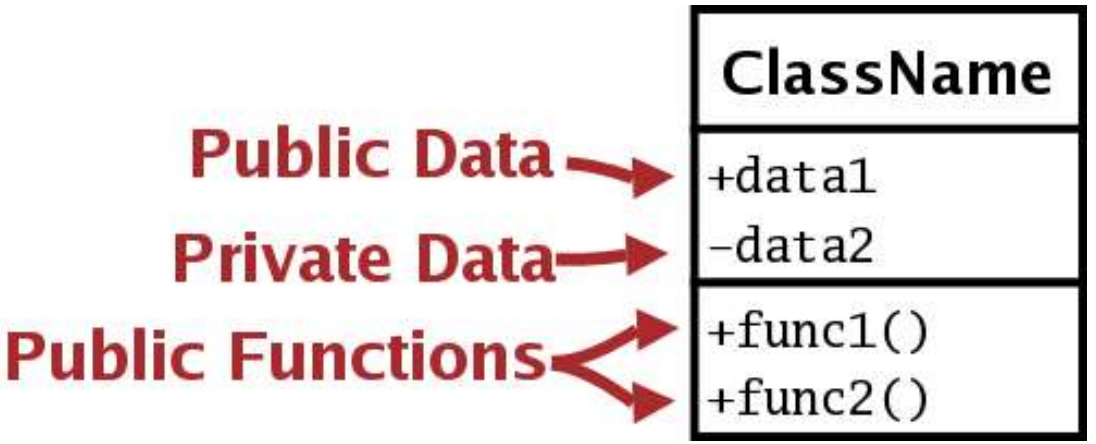
- A class diagram models the classes in a system and how they are related.
- Classes are modeled as boxes with compartments for:
  - The class name.
  - Attributes - the data members of the class.
  - Operations - the methods of the class.



Class Diagrams

(part 2 of 7) Member Visibility

- Compartments (except the name) can be omitted if not needed for the purpose of the diagram.
- Characters placed in front of class members indicate visibility:
  - **+** Public
  - **#** Protected
  - **-** Private
  - **~** Package



Class Diagrams

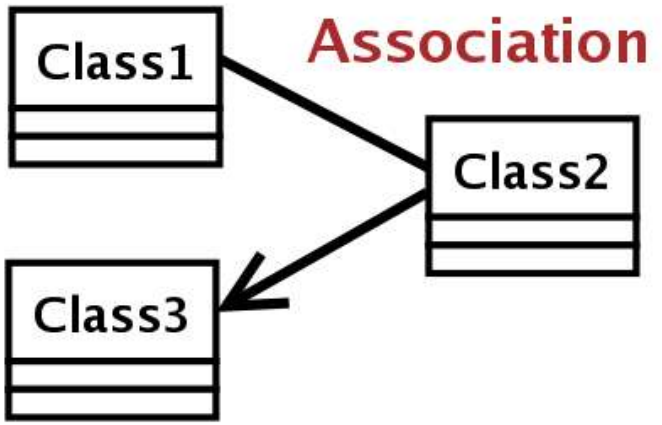
(part 3 of 7) Details

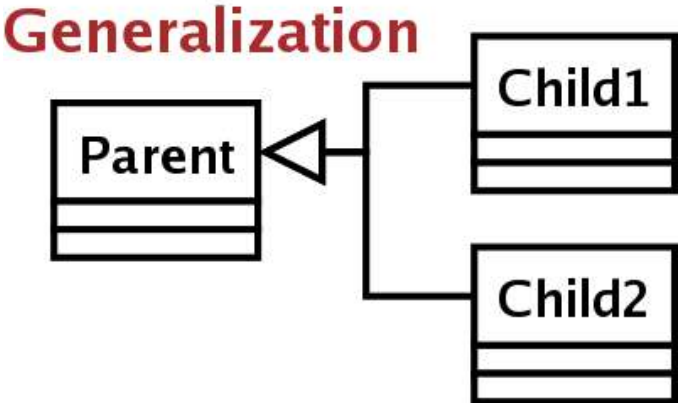
- Other class modeling details:
  - The order of the compartments is always the same: class name, attributes, and operations.
  - Members are listed in order of decreasing visibility, from public down to private.
  - Functions for getting and setting attributes are often omitted from the diagram.
  - Abstract classes are represented by having their class name in italics.
  - Pure virtual functions also have their names in italics.

Class Diagrams

(part 4 of 7) Associations and generalizations

- Many different relationships:
  - *Associations* - Arrows indicate the direction of the relation. Class1 and Class2 know about each other, and Class2 knows about Class3, but Class3 is not aware of anyone else.
  - *Generalization* - Indicates inheritance - the Parent is a generalization of the Child1 and Child2.

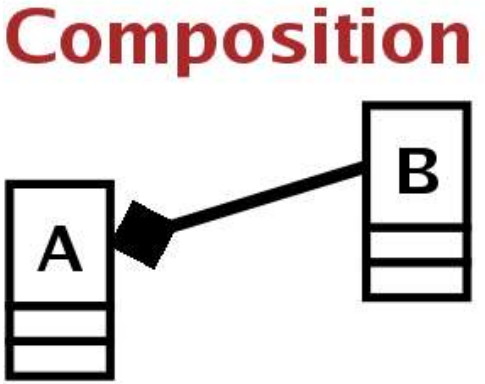


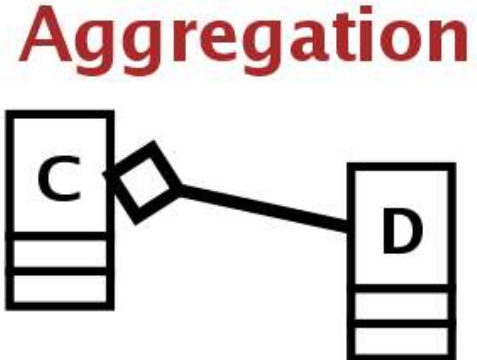


Class Diagrams

(part 5 of 7) Compositions and aggregations

- *Composition* - A is composed of Bs, like a building is composed of rooms. Usually the lifetime of B is strongly tied to the lifetime of A.
- *Aggregation* - Weaker form of composition. C has a collection of Ds, like a shopping list has a collection of items.
- Don't worry too much about getting the diamonds right - if in doubt, don't include them.

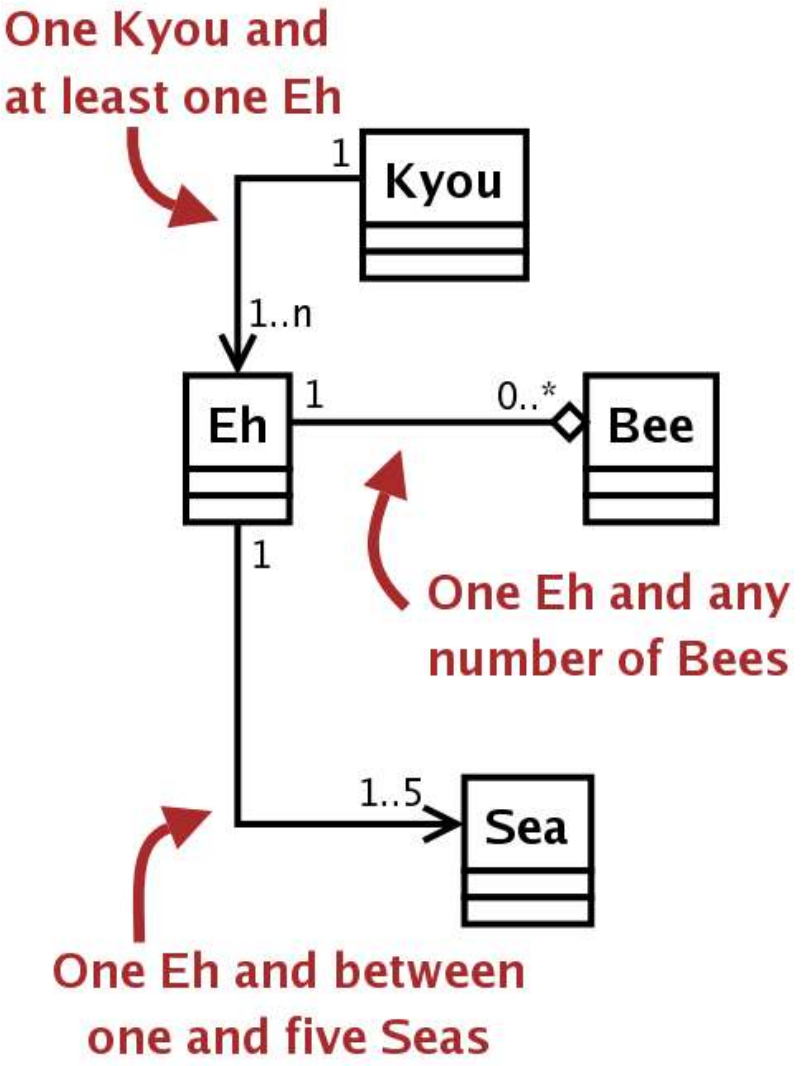




Class Diagrams

(part 6 of 7) Multiplicity

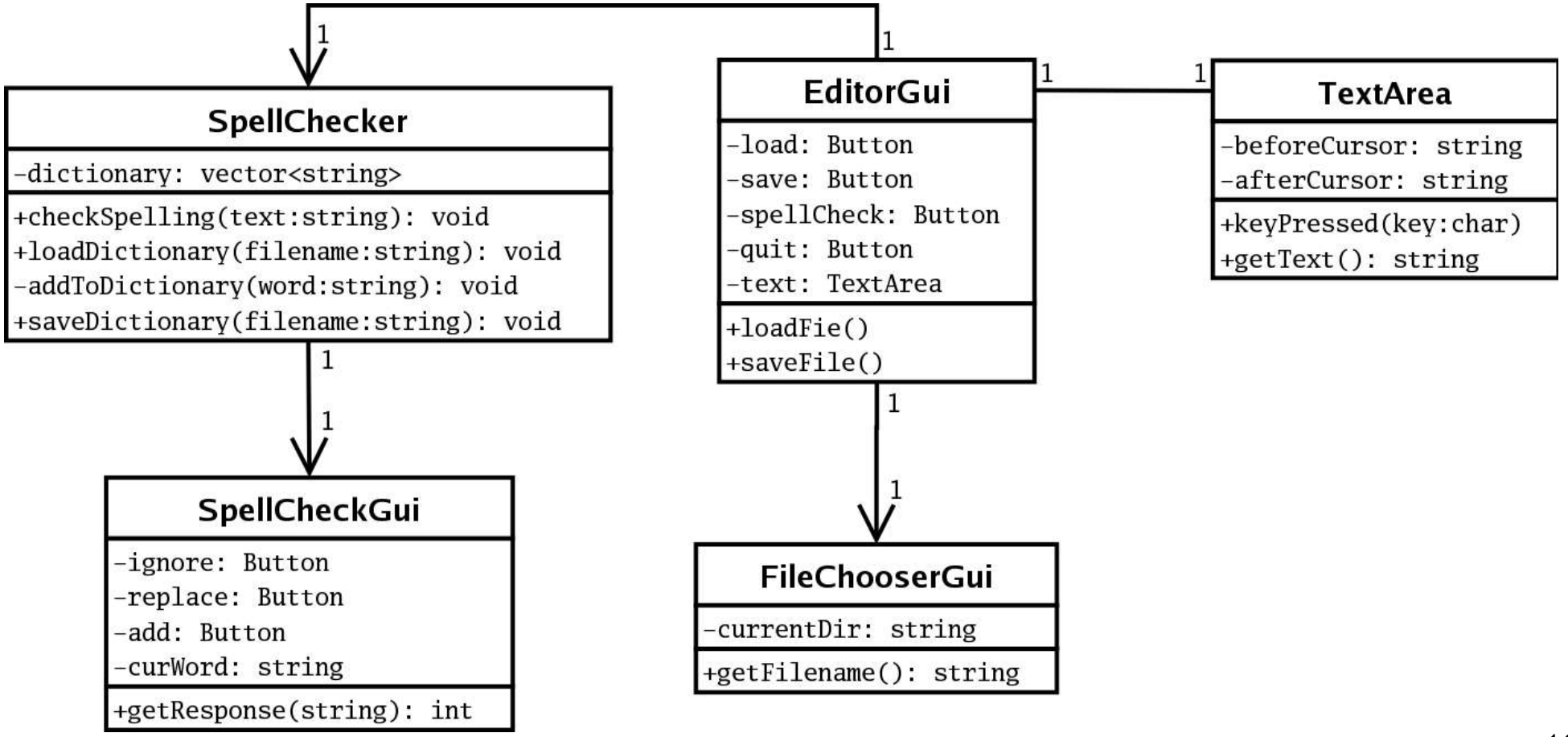
- *Multiplicity* indicates the number of instances that can be on either end of a relationship.
  - 0..1 Zero or one instance
  - 0..* Any number
  - 1 Exactly one instance
  - 1..* At least one
  - n..m General form



Class Diagrams

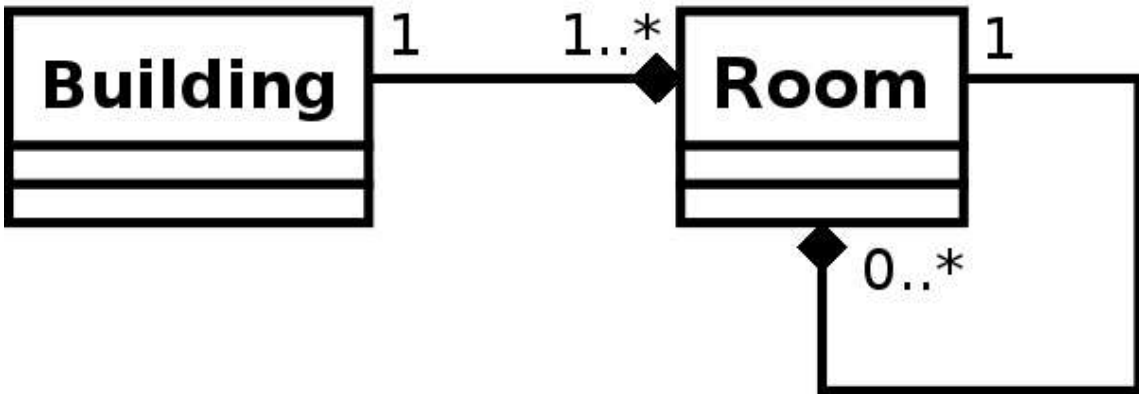
(part 7 of 7) Example

- Class diagram for a text editor:



Object Diagrams
(part 1 of 2)

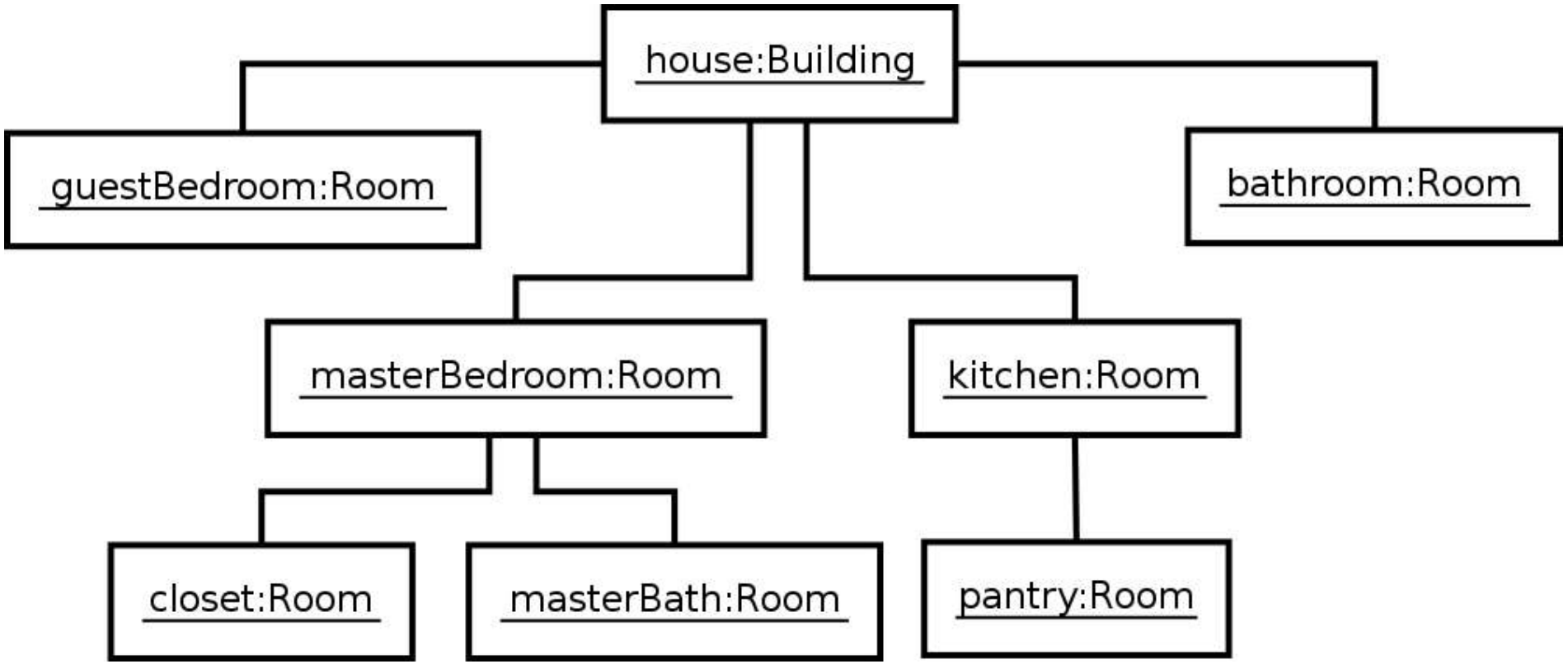
- An object diagram shows instances of classes and their relationships at a particular point in time.
- Useful for explaining complex relationships.
- Consider this small class diagram:



Object Diagrams

(part 2 of 2) Example

- An object diagram could show how instances of those classes are used to represent a house:



Sequence Diagrams

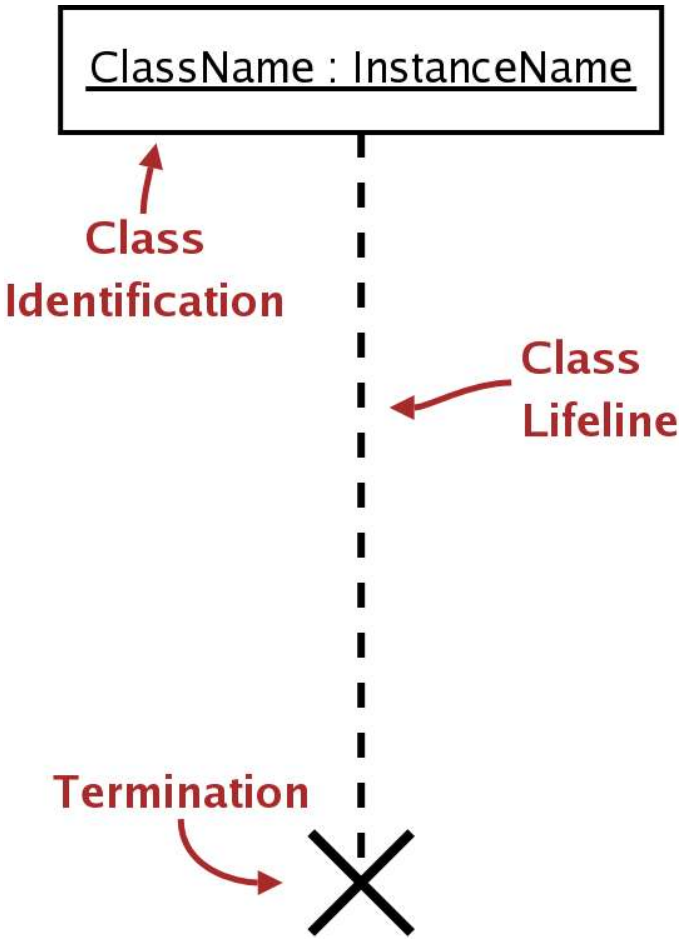
(part 1 of 5) Organization and use.

- A sequence diagram details how an operation is carried out.
  - Shows what messages are from one object to another and when they are sent.
  - Organized vertically by time - time flows down.
  - Horizontal axis shows classes or class roles.
  - Usually an individual diagram shows the sequence of events for some particular feature rather than for the whole program.

Sequence Diagrams

(part 2 of 5) Vocuabulary

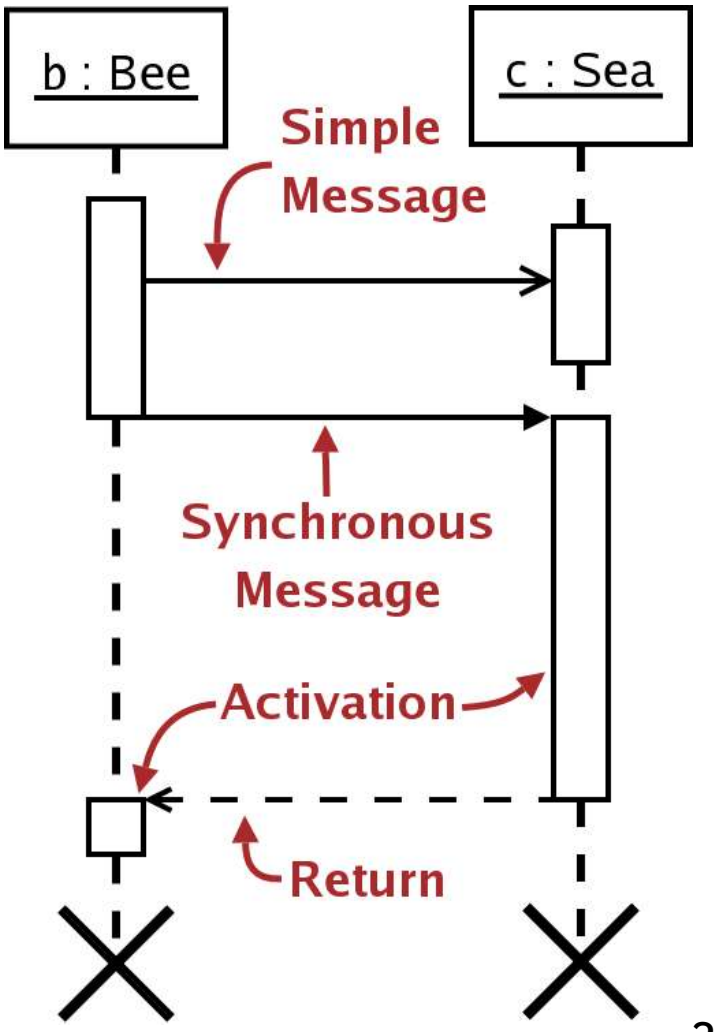
- Diagram vocabulary:
  - *Class Identification* - a box with underlined name in form of *InstanceName : ClassName*.
  - *Class Lifeline* - a dotted line indicating the object exists.
  - *Termination* - An X at the end of the lifeline indicating the object was destroyed.



Sequence Diagrams

(part 3 of 5) More vocabulary

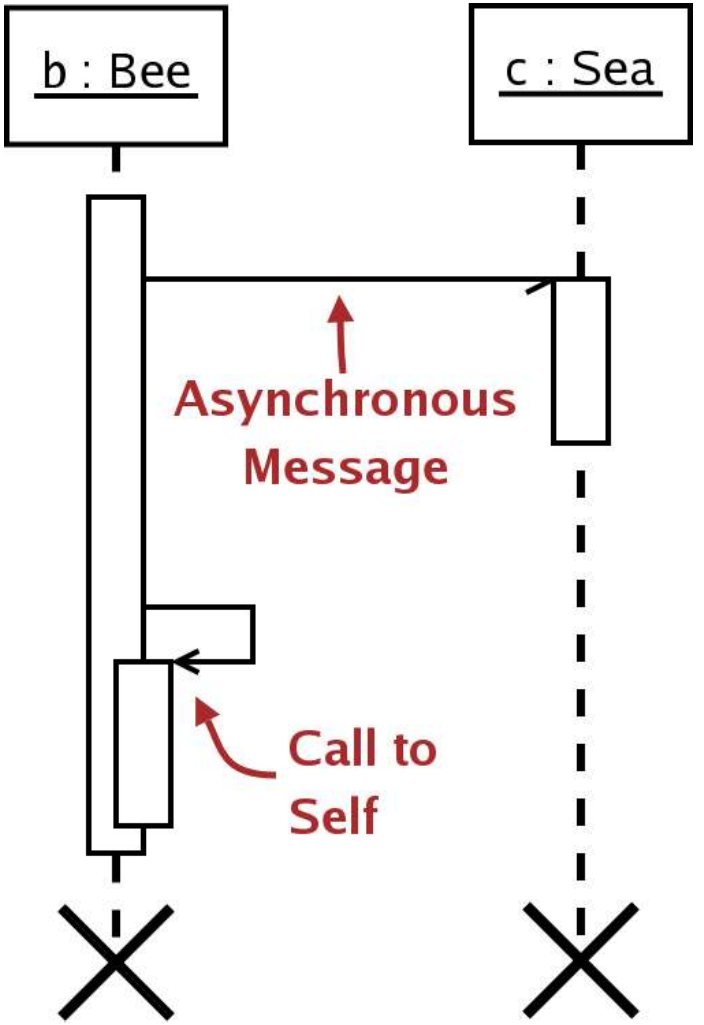
- *Activation* - A box over the lifeline indicates that class or object has control.
- *Simple message* - A line with a line arrow indicates a message or function call.
- *Syncronous message* - Indicated by a line with a filled arrow. A dashed line with an arrow in opposite direction indicates a return.



Sequence Diagrams

(part 4 of 5) Yet more vocabulary

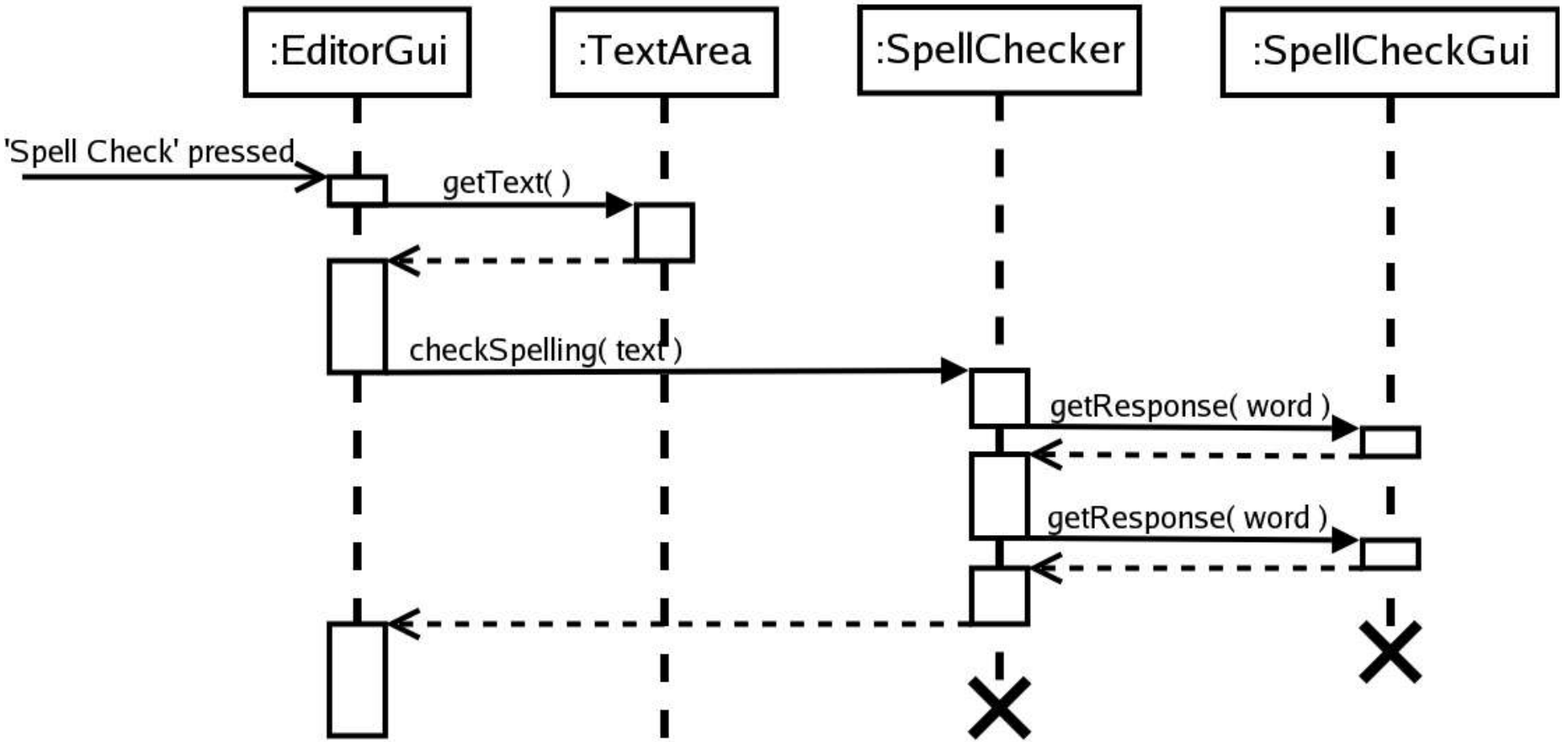
- *Asynchronous message* - A line with a half arrow indicates a message that does not stop processing in the sender.
- *Call to self* - An object calling itself is indicated by a message and a sub-activation box.
- Usually messages are labeled.



Sequence Diagrams

(part 5 of 5) Example

- Sequence diagram for text editor spell checking:



Collaboration Diagrams

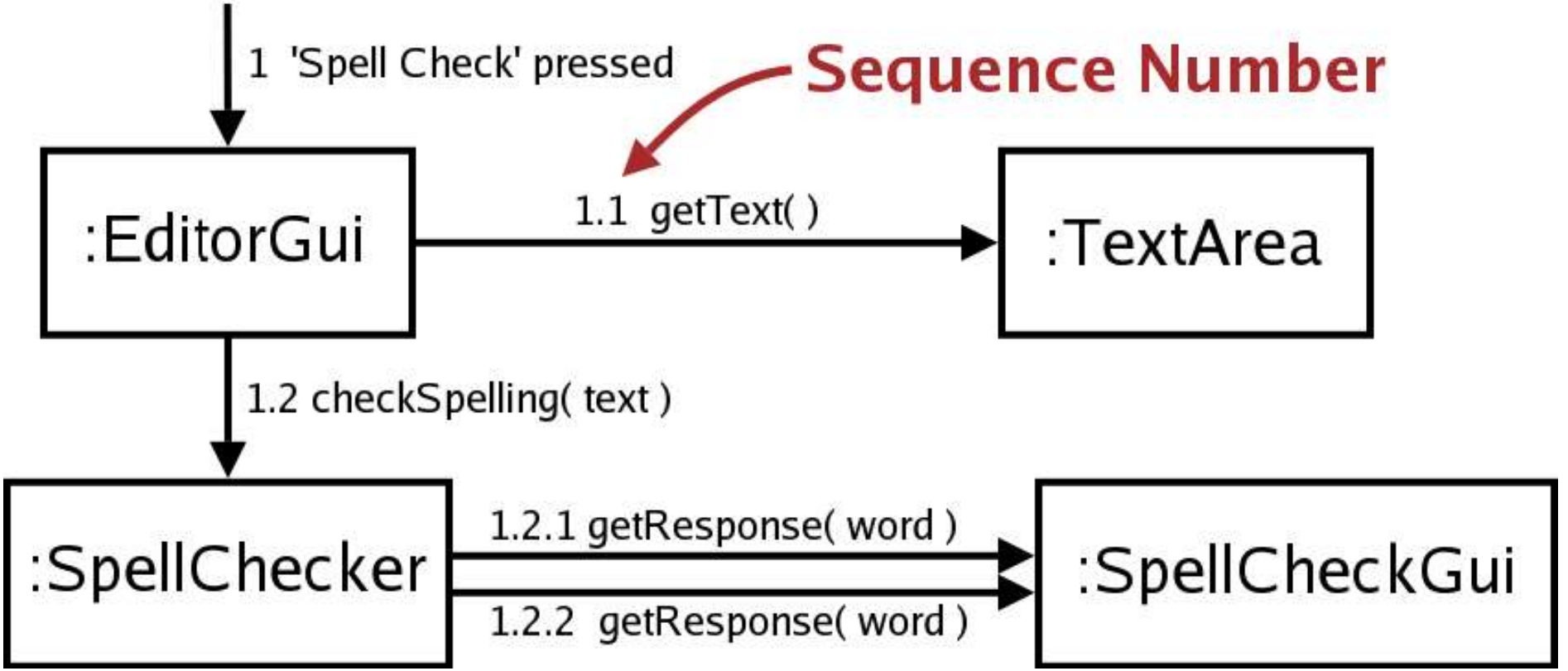
(part 1 of 2) Diagram vocabulary

- A collaboration diagram models the flow of messages between objects.
- Vocabulary is similar to sequence diagrams.
  - Classes are represented by boxes with names in the form of *instance/role name : class name*. Instance names are underlined.
  - Message types are the same as in sequence diagrams.
  - Messages have a sequence number.
  - Time is indicated by sequence numbers rather than the arrangement of the diagram.

Collaboration Diagrams

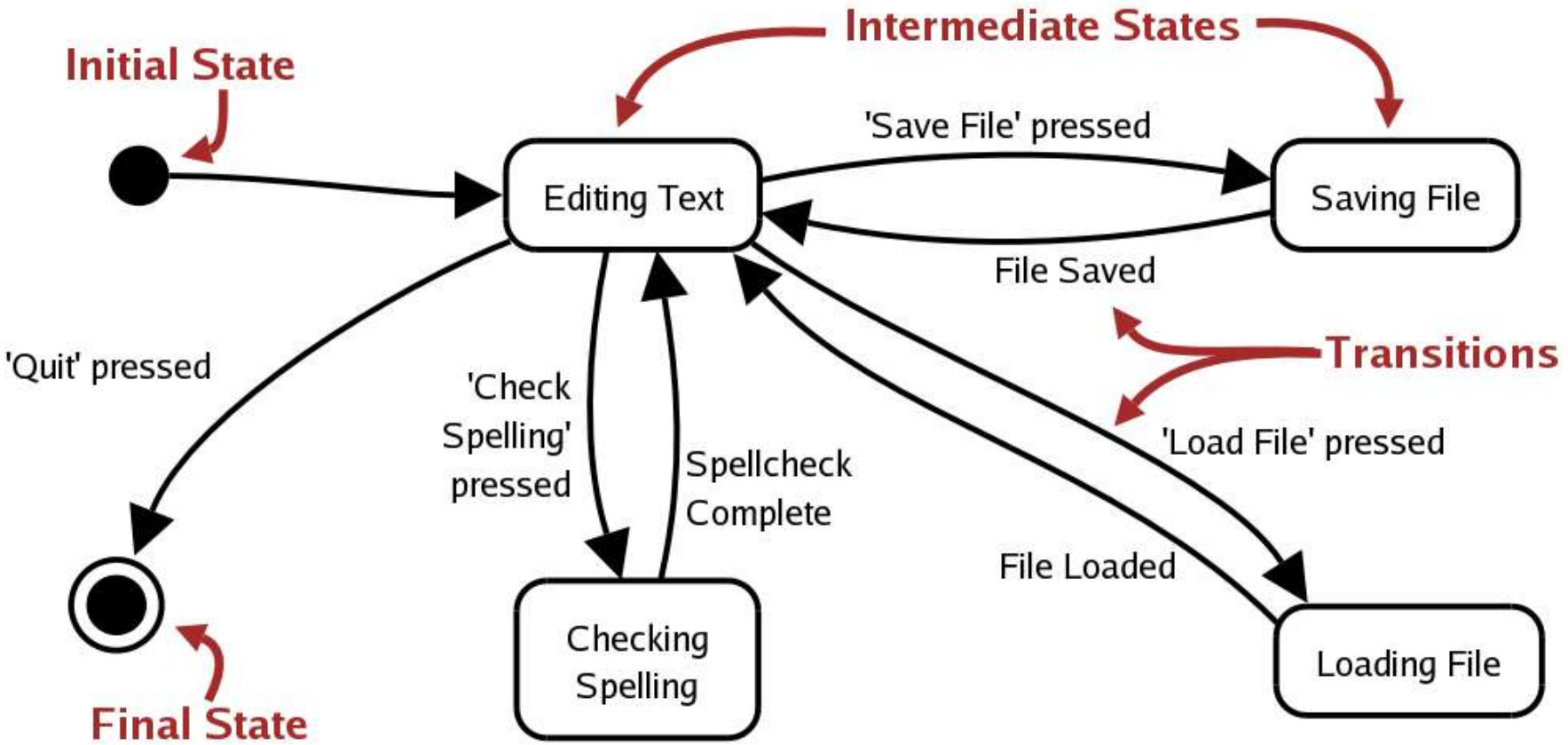
(part 2 of 2) Example

- Collaboration diagram for text editor spell checking:



Statechart Diagrams

- A statechart diagram shows the states an object can be in and the transitions between states.



Activity Diagrams

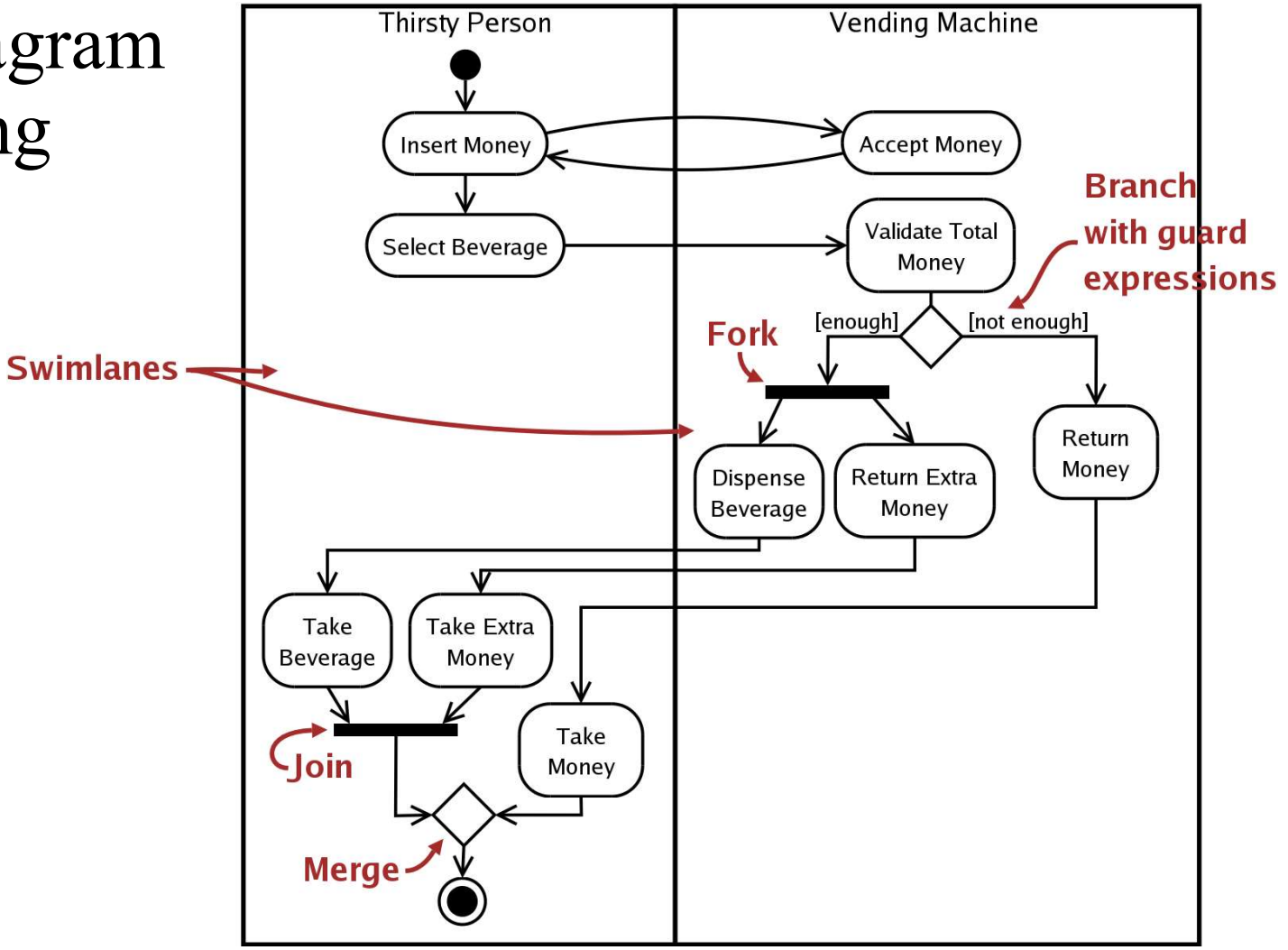
(part 1 of 2) Purpose and parts

- An activity diagram is like a flowchart.
- Shows the logic of some operation.
  - States are actions.
  - Can have multiple objects. The diagram is divided into swimlanes, one lane for each object.
  - Can have branches like a flowchart.
    - Drawn as diamonds
    - Need guard expressions to label the transitions out.
  - Can have forks and joins.

Activity Diagrams

(part 2 of 2) Example

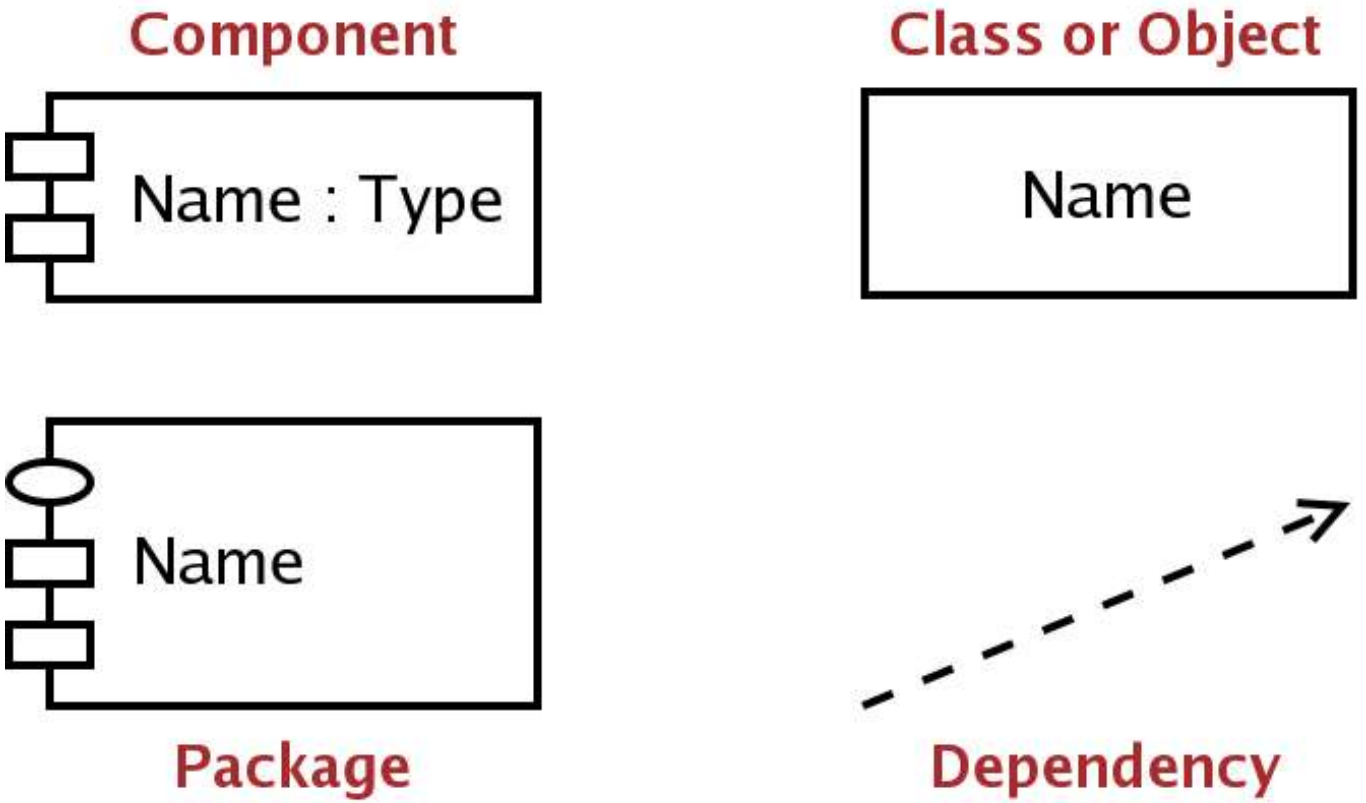
- Activity diagram for a vending machine:



Component and Deployment Diagrams

(part 1 of 2)

- A component diagram shows the relationships between the major parts of a system.



Component and Deployment Diagrams

(part 2 of 2)

- A deployment diagram shows where the components of a system are physically located.
- In addition to the vocabulary from component diagrams, a deployment diagram uses *nodes* and *communication relationships*: