

Approved	Checked	Prepared
		Hieu Tran Sept 23, 2016

Introduction to UML Diagram

Software Tool Solution 1 Group
Software Solution 3 Section
Software Engineering Division
Renesas Design Vietnam Co., Ltd.

Revision History

Date	Version	Description	Author
Sept. 23, 2016	1.0	Newly create	Hieu Tran

Table of Contents



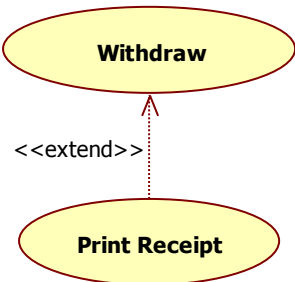
1. Use case diagram	4
1.1 Usage	4
1.2 Notation	4
1.3 Example	5
2. Class diagram	6
2.1 Usage	6
2.2 Notation	6
2.2.1 Class	6
2.2.2 Interface	7
2.2.3 Relationship	7
2.3 Example diagram	10
3. Activity Diagram	11
3.1 Usage	11
3.2 Notation	11
3.3 Example	13
4. Sequence diagram	15
4.1 Usage	15
4.2 Notation	15
4.3 Example	18
5. Reference	18

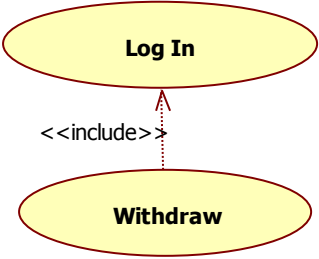
1. Use case diagram

1.1 Usage

- Use case diagrams are usually referred to as behavior diagrams used to **describe a set of actions** (use cases) that **some system or systems** (subject) should or **can perform in collaboration with one or more external users of the system** (actors).
- There are two type of use case diagram:
 - **Business Use Case Diagrams: what you see in actual work.** It represents business function, process, or activity performed in the modeled business. A business actor represents a role played by some person or system external to the modeled business, and interacting with the business.
 - **System Use Case Diagrams: output of your design, analysis.** It contains requirements (what the system is supposed to do), functionality (what the system can do).

1.2 Notation

Notation	Explanation
Actor  User	Actor is stake holder of the system. It can be a person, an external system,...
Use case 	A use case is a list of actions or event steps , typically defining the interactions between an actor and the system , to achieve a goal . The name of use case should start by a verb .
Extend Relationship  Print Receipt is optional when Withdraw	Extend is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) extending use case can be inserted into the behavior defined in the extended use case. Extend relationship is shown as a dashed line with an open arrowhead directed from the extending use case to the extended (base) use case . The arrow is labeled with the keyword <<extend>> .

Notation	Explanation
<p>Include Relationship</p>  <p>To Withdraw, you always Log In.</p>	<p>Use case include is a directed relationship between two use cases which is used to show that behavior of the included use case (the addition) is inserted into the behavior of the including (the base) use case.</p> <p>Include relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case. The arrow is labeled with the keyword <<include>>.</p>

1.3 Example

Below is simple example system use case diagram for an ATM Transaction:

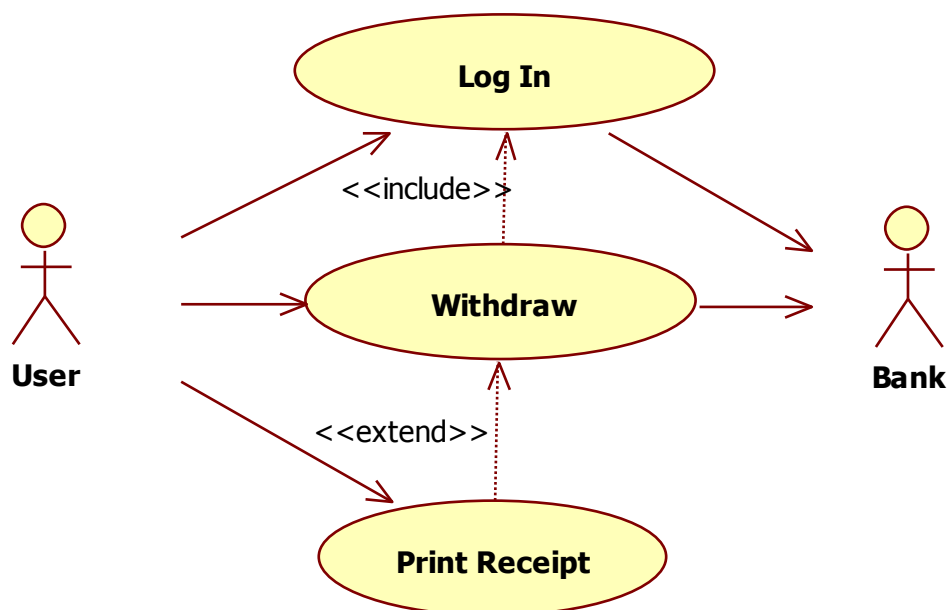


Figure 1: Sample use cases for ATM Transaction

Explanation (user story):

- A user can do below activity with the ATM system:
 - User can log in to his account. Then the ATM connects to Bank system to authenticate the login.
 - User can withdraw his account. Of course he must login to ATM system. User can print receipt for withdraw transaction.

2. Class diagram

2.1 Usage

- Shows structure of the designed system at the level of classes and interfaces
- Shows their features, constraints and relationships - associations, generalizations, dependencies, etc.

2.2 Notation

2.2.1 Class

- Class notation has are three main parts:

Class name
Class attributes
Class methods

- Notation for declare attribute, method:

	Notation	Example
Attribute	Name: Type	EmployeeCode: int
Method	Name(parameter: Type, ...): Type	DoSomething(when: string, time: int): bool

- Notation for visibility:

	Notation	Example
Public	Start with “+”	+PublicAttribute: int +PublicMethod(): int
Protected	Start with “#”	#ProtectedAttribute: int #ProtectedMethod(): int
Private	Start with “-“	-privateAttribute: int -privateMethod(): int
Package (Java) Internal (C#)	Start with “~”	~InternalAttribute: int ~InternalMethod(): int
Abstract (for class, method)	<i>Italic</i>	<i>AbstractClass</i> <i>AbstractMethod(): int</i>

	Notation	Example
Static (for attribute, method)	<u>Underline</u>	<u>+PublicAttribute: int</u> <u>+PublicMethod(): int</u>

2.2.2 Interface

- An interface may be shown using a rectangle symbol with the keyword «interface» preceding the name. The declaration of attributes, methods are same as class:



Figure 2.1: Interface declaration

- Interface participating in the interface realization dependency is shown as a circle or ball, labeled with the name of the interface:



Interface

Figure 2.2: Interface in the interface realization dependency

2.2.3 Relationship

2.2.3.1 Generalization / Inheritance (is-a)

- A generalization is shown as a line with a hollow triangle as an arrowhead between the symbols representing the involved classifiers. The arrowhead points to the symbol representing the general classifier:

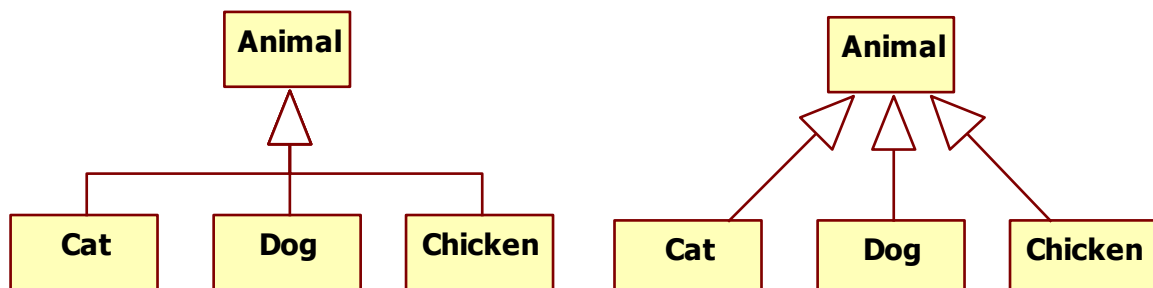


Figure 2.3: Class Inheritance

- Generalization relationship is also informally called "Is A" relationship. For example in above Figure: "Cat is an Animal"...
- For Inheritance from Abstract class or interface, we use below notation:

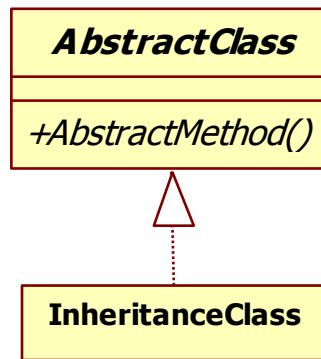


Figure 2.4: Class inherit an Abstract class

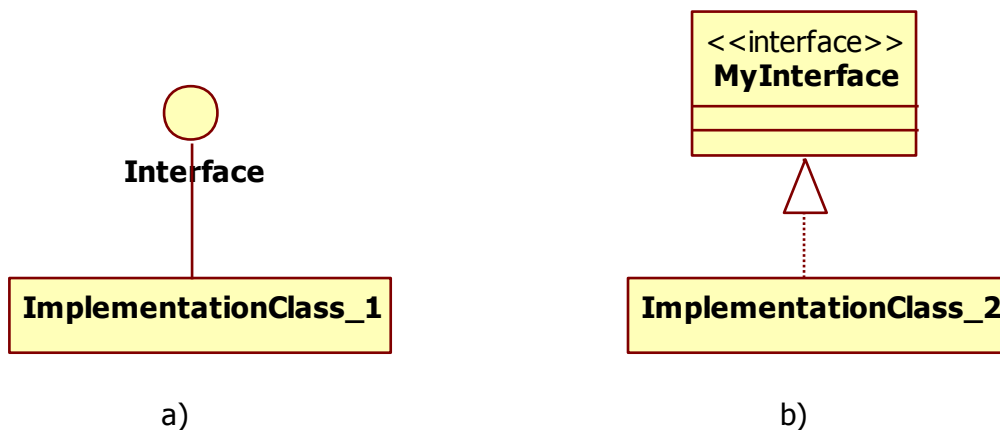


Figure 2.5: Class implement Interface

2.2.3.2 Association

- Association is a relationship between classifiers which is used to show that instances of classifiers could be either linked to each other or combined logically or physically into some aggregation.

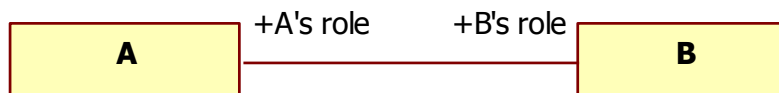


Figure 2.6: Association between A and B.

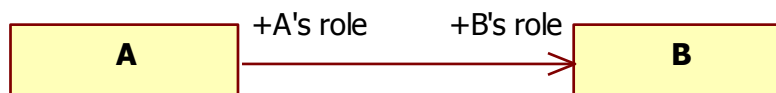


Figure 2.7: B is navigable from A.

2.2.3.3 Dependency

- Dependency is a relationship which show that some UML element or a set of elements requires, **needs or depends on** other model elements **for specification or implementation**.







Figure 8: Dependency association between Garage and Car

Explanation: Garage class needs Car class to implement its methods. For example, if Car class is not specified, Garage cannot implement the methods which return object whose type is Car.

2.2.3.4 Aggregation and Composition (has-a)

Aggregation and Composition are forms of association.

	Aggregation	Composition
Main difference	<ul style="list-style-type: none"> - Shared part could be included in several composites. - If some or all of the composites are deleted, shared part may still exist. 	<ul style="list-style-type: none"> - A part could be included in at most one composite at a time. - If a composite is deleted, all of its composite parts are "normally" deleted with it.
Notation		
Example	<p>Associtaion between Triagle - Edge:</p>  <ul style="list-style-type: none"> - An edge can be contained in many triangles - When triangles contain an edge are deleted, the edge can be or not deleted. 	<p>Association between Folder - File:</p>  <ul style="list-style-type: none"> - A file can be only contained in only one folder - When folder is deleted, files in this folder are also deleted

- Aggregation and Composition are also informally called "Has A" relationship. For example: "Triangle has an Edge", "Folder has a File"...

2.2.3.5 Multiplicity in Aggregation, Composition, or Association

- Multiplicity notation:

Multiplicity	Cardinality
*	Any number
1	Exactly 1
n	Exactly n
0..1	Zero or one
1..*	One or more
m..n	m through n

- Notation:



Figure 2.9: Multiplicity between A and B

- Explanation for above figure:
 - Each A is associated with any number of B.
 - Each B is associated with exactly one A.

2.3 Example diagram

Below is sample class diagram:

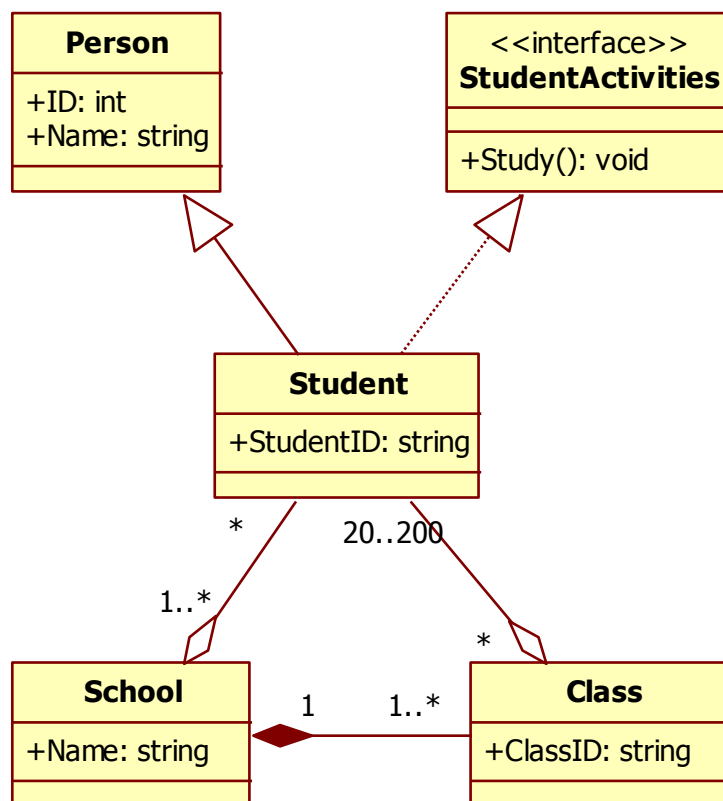


Figure 10: Sample Class Diagram

Explanation in natural language:

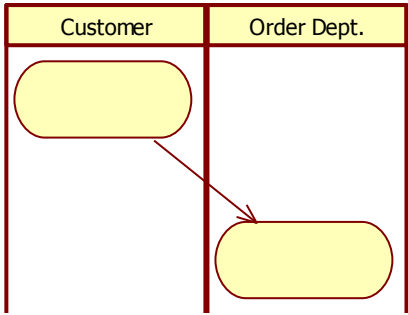
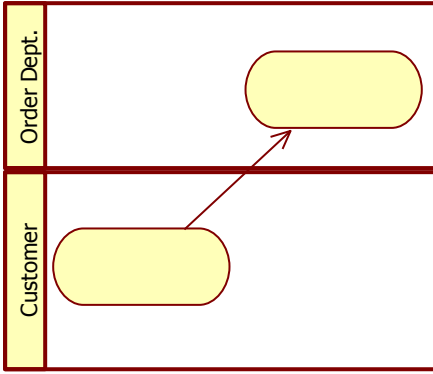


- A school can has many student and classes. However, a class is only belong to a specific school.
- Student is a person which has a student ID beside the common ID and Name. A student has specific activities such as study... Student can join in many classes in one or many school.


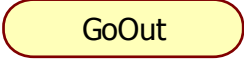



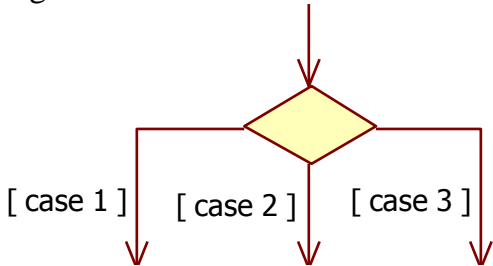
3. Activity Diagram

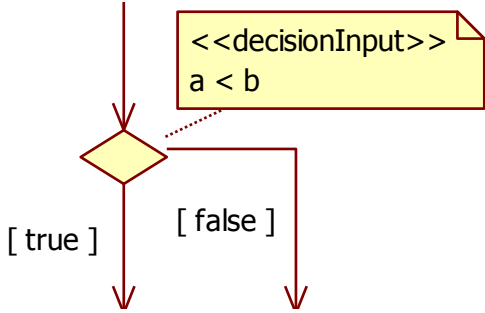
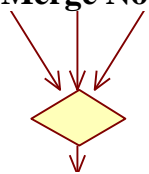
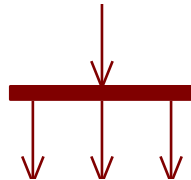
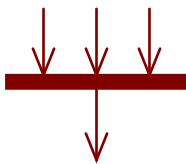
3.1 Usage

- Shows flow of control or object flow with emphasis on the sequence and conditions of the flow.

3.2 Notation

Notation	Explanation
<p>Swimlane</p> <p>a. Horizontal swimlanes</p>  <p>b. Vertical swimlanes</p> 	<p>Partitions often correspond to organizational units or business actors in a business model and may be shown using swimlane notation.</p>
<p>Activity initial node</p> 	<p>Initial node is a control node at which flow starts when the activity is invoked.</p> <p>Activities may have more than one initial node. In this case, invoking the activity starts multiple flows, one at each initial node.</p>
<p>Flow Final Node</p> 	<p>Flow final node is a control final node that terminates a flow.</p>
<p>Activity Final Node</p>	<p>Activity final node is a control final node that stops all flows in an activity</p>

Notation	Explanation
	
Action 	Action is a named element which represents a single atomic step within activity
Send Signal Action 	Send signal action is an invocation action that creates a signal from its inputs, and transmits it to the specified target object. The sender of the signal (aka "requestor") continues execution immediately, without waiting for any response.
Accept Signal Action 	Accept signal action is action that waits for a specific event.
Object 	An object node is an abstract activity node that is used to define object flow in an activity.
Decision Node a. Decision node with three outgoing edges  b. Decision node with decision input behavior.	Decision node is a control node that accepts tokens on one or two incoming edges and selects one outgoing edge from one or more outgoing flows.

Notation	Explanation
	
Merge Node 	<p>Merge node is a control node that brings together multiple incoming alternate flows to accept single outgoing flow.</p> <p>Merge node does not synchronize incoming concurrent flows.</p>
Fork Node 	<p>Fork node is a control node that has one incoming edge and multiple outgoing edges and is used to split incoming flow into multiple concurrent flows. Fork nodes are introduced to support parallelism in activities.</p>
Join Node 	<p>Join node is a control node that has multiple incoming edges and one outgoing edge and is used to synchronize incoming concurrent flows.</p>

3.3 Example

Below is a sample for activity diagram

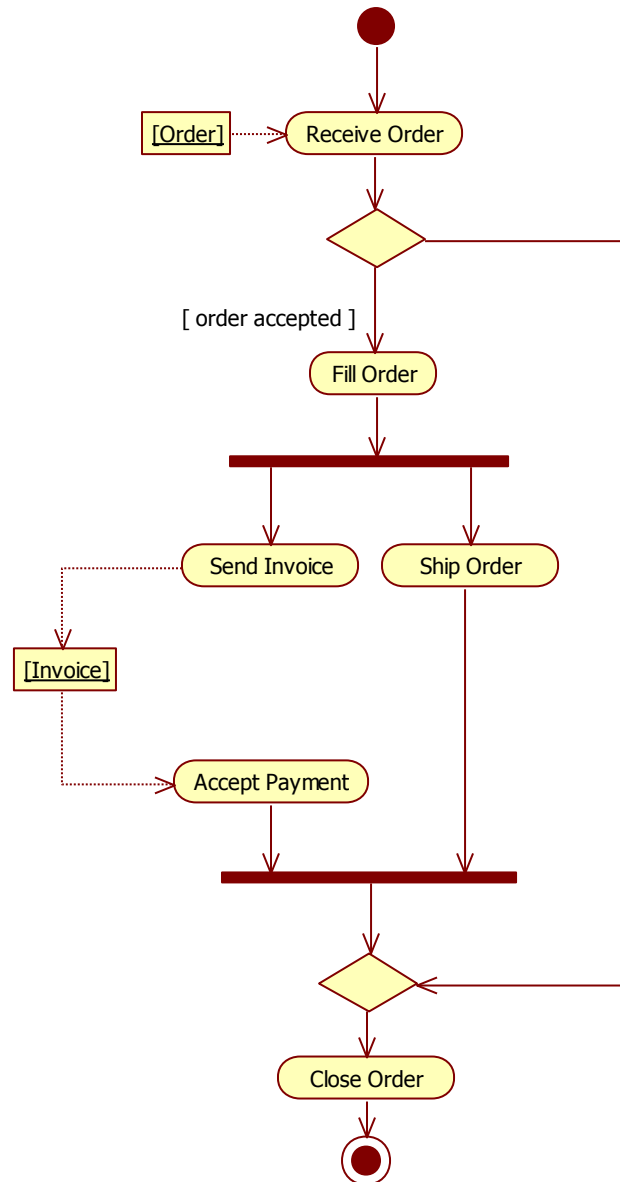


Figure 11: Sample activity diagram for a “Process Order”

Explanation for “Process Order” activity of a sale-man:

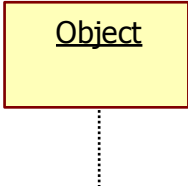
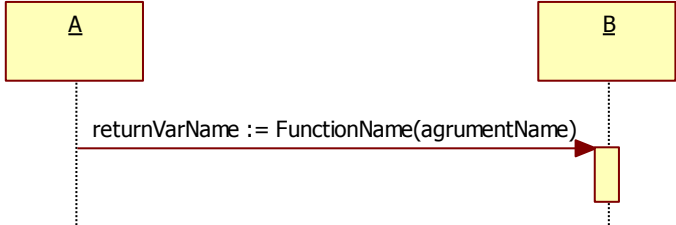
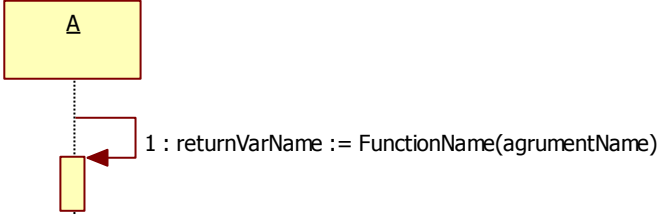
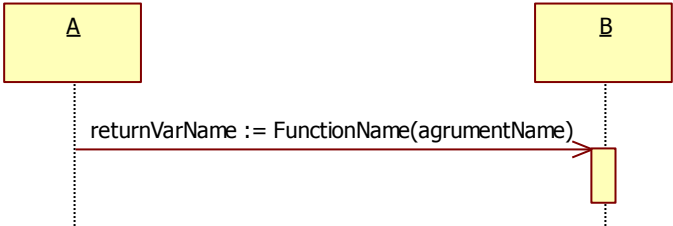
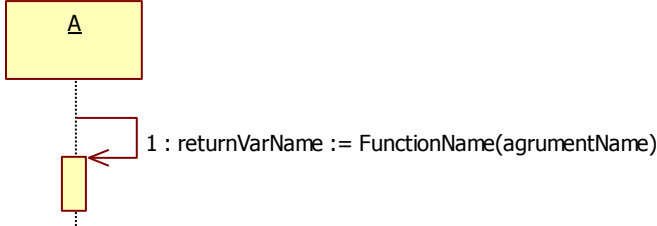
- First, the sale-man receives the order from customer. Then he decides to accept this order or not. If the order is reject, he changes order status to close. Otherwise, when order is accepted, he fills the order. Then he ships the goods and sends invoice and receives payment at the same time. After these action done, the order is also closed.

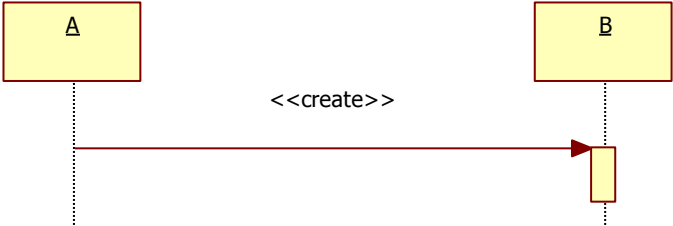
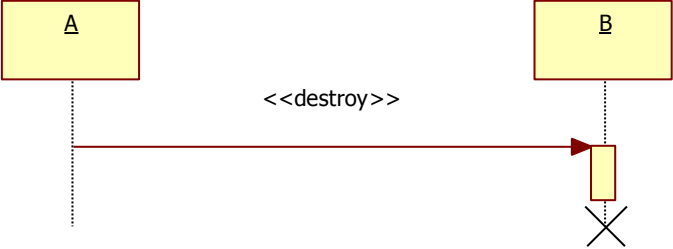
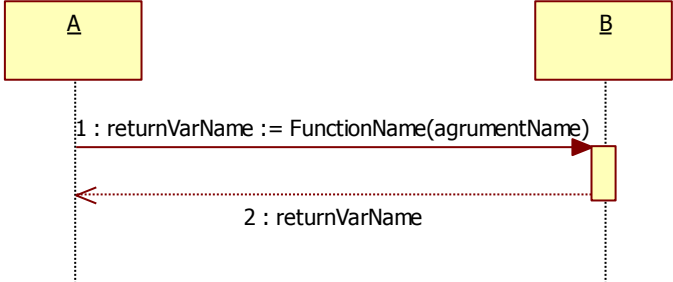
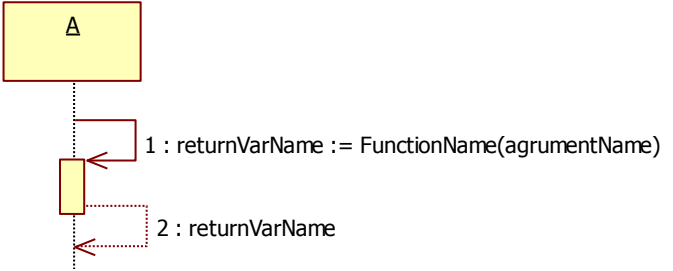
4. Sequence diagram

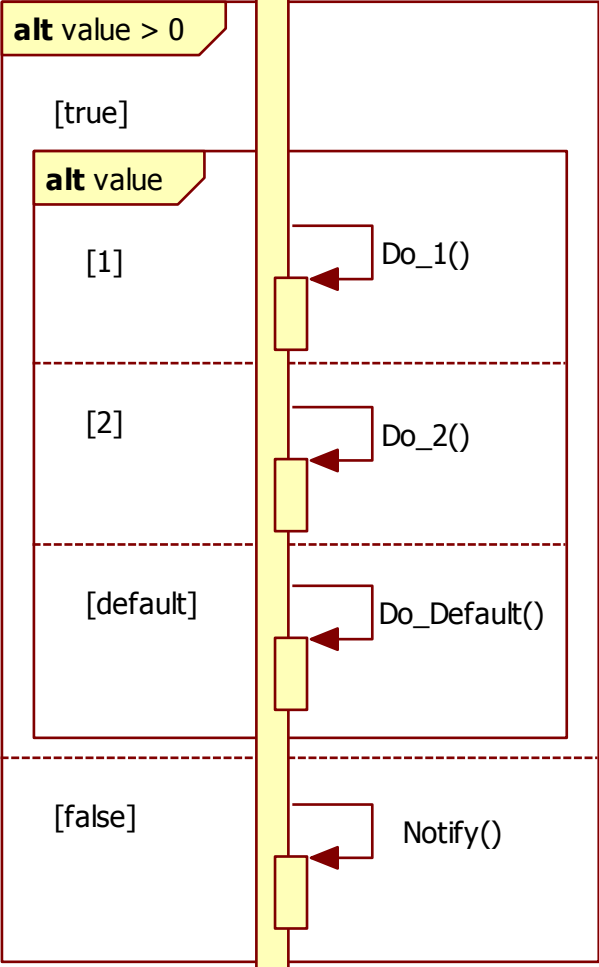
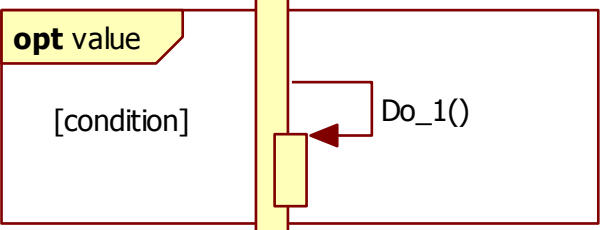
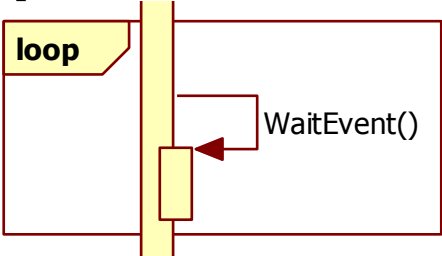
4.1 Usage

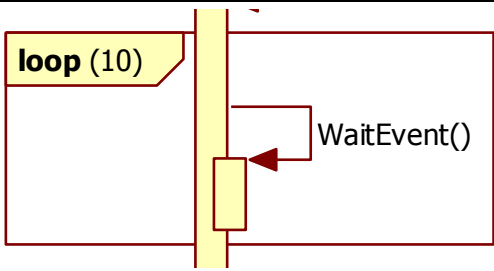
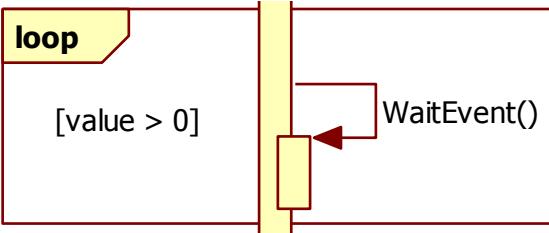
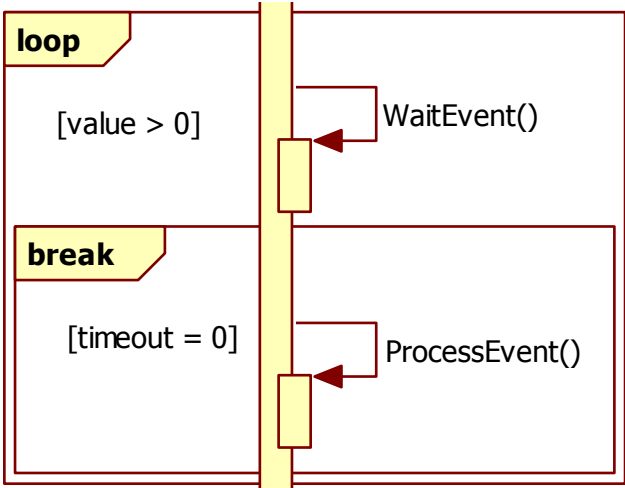
- Describes interactions by focusing on the sequence of messages that are exchanged among participants.

4.2 Notation

Notation	Explanation
Lifeline 	Time line of an object
Message	
Synchronous Call 	A call method "FunctionName" of B and wait for return result
Synchronous Self-Call 	A self-call method "FunctionName" and wait for return result
Asynchronous Call 	A call method "FunctionName" of B without wait for return result
Asynchronous Self-Call 	A self-call method "FunctionName" without wait for return result

Notation	Explanation
Constructor Call  <pre> sequenceDiagram participant A participant B A->>B: <<create>> activate B deactivate B </pre>	A call constructor of B
Destructor Call  <pre> sequenceDiagram participant A participant B A->>B: <<destroy>> activate B destroy B </pre>	A call destructor of B
Return message  <pre> sequenceDiagram participant A participant B A->>B: 1 : returnVarName := FunctionName(agrumentName) activate B B-->>A: 2 : returnVarName deactivate B </pre>	B return value of “FunctionName” called by A
Self-return message  <pre> sequenceDiagram participant A A->>A: 1 : returnVarName := FunctionName(agrumentName) activate A A-->>A: 2 : returnVarName deactivate A </pre>	A self-return value of “FunctionName”
Combined Fragment	
Alternatives	Use for if-else or switch statement

Notation	Explanation
	
<p>Option</p> 	<p>Use for if statement without else statement</p>
<p>Loop</p>  <p>a.</p>	<p>a. Infinite loop.</p> <p>b. Loop to execute exactly 10 times.</p>

Notation	Explanation
<p>b.</p>  <p>c.</p> 	<p>c. Loop with condition</p>
<p>Break</p> 	<p>Use to Break in Loop statement</p>

4.3 Example

T.B.D

5. Reference

- Use case diagram: <http://www.uml-diagrams.org/use-case-diagrams.html>
- Class diagram: <http://www.uml-diagrams.org/class-diagrams-overview.html>
- Activity diagram: <http://www.uml-diagrams.org/activity-diagrams.html>
- Sequence diagram: <http://www.uml-diagrams.org/sequence-diagrams.html>