UML Class Diagram

UML Class Diagram Example

```
<<persistent>>
Project

+ pid: char
+ location: string = "Phoenix"
+ cost: real

<<create>>
+ newProject();

<<destroy>>
+ removeProject();

<<standardAccess>>
+ getPID();
+ # setPID();
+ getLocation();
+ # setLocation(string location);
+ getCost();
+ # setCost(real cost);
```

UML Class Diagram 2
Details

Visibility
+publicAttribute, -privateAttribute, #protectedAttribute

Stereotype
<<persistent>>, <<create>>, …

Attribute syntax
[visibility] name [multiplicity] [: type] [= initial-value] [{property-string}]

Operation syntax
[visibility] name [(parameter-list)] [: return-type] [{property-string}]
where each parameter has the form
[direction] name : type [= default-value]
and directions can be - in, out, inout.

Connecting classes

There are 3 types of connections in UML (Inter-class relationships)

- **Generalizations** which are *inheritance* relationships.
- **Associations** which are structural relationships: Represents relationships between instances of two classes.
  - Aggregation and Composition
- **Dependencies** which are *using* relationships.
**Relationships: 3 Kinds**

- **Window**
  - open()
  - close()

- **Event**

- **ConsoleWindow**
- **DialogBox**
- **Control**

**Generalization**

- Relationship between general thing (parent) and more specific thing (child)
- Child “is-a-kind-of” parent.
- Child inherits attributes and operations of parent.
- Use when you want to show parent child relationship
**Associations (UML)**

- Represent conceptual relationships between classes (structural)
  - Customers make Orders
  - Professors teach Courses

**relationship name**

**direction indicator:** how to read relation name

**Multiplicity**
defines the number of objects associated with an instance of the association.
Default of 1:
Zero or more (*);
n..m; range from n to m inclusive

Sometimes **directed arrows** are used to specify navigability.

---

**Associations - In Other OOAD**

Associations may be binary, ternary, or higher order

**binary association**

**Ternary association**
**Associations – UML Links**

- Link is a semantic connection among objects.
- A link is an instance of an association.

```plaintext
Company

1..* employee

assign(development)

Company

* employer

+setSalary(s: Salary)
+setDept(d: Dept)

w: Worker

Named object

Worker

Anonymous object

link
```

**Associations – Link Attributes**

- **Link Attributes**
  
  The most compelling reason for having links and attributes is for-
  many-to-many relationships

```
File

User

access permission

link attribute
```
**Associations – Link Attributes**

- **UML Association Class**
- An association class is used when a relationship between objects has *properties*.

**Associations - Aggregation**

- Structural association representing “whole/part” relationship.
- “has-a” relationship.
Aggregation and Composition

Composition is a stronger form of aggregation. Composite parts live and die with the whole.

Dependency

- A change in one thing may affect another.
- "Uses" relationship.
- May have a name, but not common.
- Use Dependency to show one thing uses another.
**Modeling Structural Relationships**

- Considering a bunch of classes and their association relationships

![UML class diagram]

**Dependencies**

A dependency shows that one class uses another. A change in one will affect the other.

```plaintext
Account
- balance: Money
- accountHolder: String
- interestRate: int

Money

- method parameters
```

Money
Abstract classes, Types, Interfaces

- **«type» SomeType**
  Types have no implementation
  Used for built in types like int.

- **«interface» SomeFace**
  Interfaces only have operations
  Java’s interfaces have attributes too.

- **{abstract} SomeClass**
  Abstract classes have no instances
  Strictly speaking, these map to Java interfaces.

Interfaces and Realization

- **Frame**
  realization
  ActionListener
  action performed()

- **Frame**
  ActionListener
Abstract classes

- **Shape**
  - `d`: Dimension
  - `draw()`

  - **Circle**
    - `draw()`

  - **Square**
    - `draw()`

Generics (Parameterized Types)

- **Priority Queue**
  - `insert(T)`
  - `remove(T)`

- **Print Queue**

  - «bind»
  - `<Print Job>`

  - a realization
    - (part dependency and part generalization)
**Association qualifiers**

- A *qualifier* is an association attribute that is unique within a set of related objects.
- A qualifier is used within a *qualified association* to relate a *qualified object* to a *target object* using a *qualifier value* from a set of qualifier values.

```
+----------------+     +----------------+     +----------------+
| Show           |     | Sales           |     | Ticket          |
| performance: Date |     | seat: Seatnumber |     |                |
+----------------+     +----------------+     +----------------+
```

**Recursive associations**

```
+-----------------+   +-----------------+   +-----------------+
| previous        |   | 0..1             |   | next             |
| Queue Element   |   | 0..1             |   | next             |
```

UML class diagrams 21

UML class diagrams 22

UML class diagrams 22
Packages

- A package is a general-purpose mechanism for organising elements into groups.
- Packages may contain other packages.
- Dependencies between packages can be shown.
- There are visibility rules for package components.
### Summary:
**Structural Modeling: Core Relationships**

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>association</td>
<td>a relationship between two or more classifiers that involves connections among their instances.</td>
<td></td>
</tr>
<tr>
<td>aggregation</td>
<td>A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.</td>
<td></td>
</tr>
<tr>
<td>generalization</td>
<td>a taxonomic relationship between a more general and a more specific element.</td>
<td></td>
</tr>
<tr>
<td>dependency</td>
<td>a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).</td>
<td></td>
</tr>
</tbody>
</table>

Reference: OMG tutorial on UML by Cris Kobryn

---

### Structural Modeling: Core Relationships (cont’d)

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>realization</td>
<td>a relationship between a specification and its implementation.</td>
<td></td>
</tr>
</tbody>
</table>

Reference: OMG tutorial on UML by Cris Kobryn
UML Interaction Diagrams

UML Behaviour diagrams

- Statechart diagram
- Interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams
Interaction Diagrams

“An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.”

- A **sequence diagram** is an interaction diagram which emphasizes the time ordering of messages between the objects.
- A **collaboration diagram** is an interaction diagram that emphasizes the structural organization of the participating objects.

Sequence diagram overview

- **Objects**
  - `c`: Client
  - `p`: ODBCProxy

- **Time**
  - `«create»`
  - `setActions(a, d, o)`
  - `setValues(d, 3.4)`
  - `setValues(a, "CO")`
  - `committed`
  - `«destroy»`

- **Object destruction**

- **Message**
  - Optional return

- **Focus of Control**
  - `setValues(d, 3.4)`
  - `setValues(a, "CO")`

- **Object Lifeline**
Collaboration diagram overview

```
c: Client

1: «create»
2: setActions(a, d, o)
3: «destroy»

«local»
.Transaction  «global»
(transient)  p: ODBCProxy

2.1: setValues(d, 3.4)
2.2: setValues(a, "CO")
```

Sequence diagram details 1

Recursion

```
:Object
```

Iteration

```
:Thing

*[i:=1..2]doStuff()
```

:Object
Sequence diagram details 2

Guards

Selection

Sequence diagram details 3

Actor

:Customer

n := getName()

new Account(n)

:Account

:Credit

creditCheck()

parameter

Elapsed

Time

UML class diagrams 33

UML class diagrams 34
**Sequence diagrams**

![Sequence Diagram]

**Sequence diagram shapes**

**Fork - centralized**
- operations can change order
- new operations may be added

**Stair - decentralised**
- Operations have strong connections
- performed in same order
- behaviour is encapsulated

---

18
What are they for

- Interaction diagrams are used to realize use-cases.
- Interaction diagrams are used to explain the behaviour of an operation.
- Interaction diagrams are used to describe the behaviour of a class.
- Interaction diagrams are used to explain collaborations among classes.
- Interactions can express appropriate levels of detail.

Use case diagrams

![Use case diagram example](image)