<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Setup</td>
</tr>
<tr>
<td>2</td>
<td>Loading yEd palettes</td>
</tr>
<tr>
<td>3</td>
<td>Comment (note symbol)</td>
</tr>
<tr>
<td>4</td>
<td>Activity Diagram</td>
</tr>
<tr>
<td>5</td>
<td>Class Diagram</td>
</tr>
</tbody>
</table>
Examples of various types of UML diagrams created using yEd graph editor.
CHAPTER 1

Setup

Installation

yEd can be downloaded and installed from official site:

- Downloads

Line intersection

By default line intersection is not marked:
To change it go to File -> Preferences:
Choose **Display:**
Change Bridge Style:
1.2. Line intersection
Loading yEd palettes

Download

To load palettes download/clone this repo with `palettes` directory:

```
> git clone https://github.com/ruslo/yed-uml
> cd yed-uml
[yed-uml]> ls -d palettes
palettes/
```

Loading into yEd

Follow these steps for each palette to load them all into yEd.

Go to `Edit -> Manage Palette...:`
Click **Import Section**:
Find palettes directory and load file:
Move palette up so it will be easier to access it:
After palette reach the top click Close:
New palette is loaded and can be used:
2.2. Loading into yEd
7.2.4 Notation

A Comment is shown as a rectangle with the upper right corner bent (this is also known as a “note symbol”). The rectangle contains the body of the Comment. The connection to each annotatedElement is shown by a separate dashed line. The dashed line connecting the note symbol to the annotatedElement(s) may be suppressed if it is clear from the context, or not important in this diagram.
CHAPTER 4

Activity Diagram

Actions

15.2.4 Notation
The notations for ActivityNodes are illustrated below. This notation is discussed in more detail in the following sub
clauses (and in Clause 16 for Actions).

16.2.4.1 Actions
Actions are notated as round-cornered rectangles, as shown in Figure 16.2. The name of the action or other description
of it may appear in the symbol.

Activity Edge

15.2.4 Notation
An ActivityEdge (whether a ControlFlow or ObjectFlow) is notated by an open arrowhead line connecting two ActivityNodes. If the edge has a name, it is notated near the arrow. Guards are shown as text in square brackets near tail of
the line.
Decision Nodes

15.3.4.3 Merge Nodes and Decision Nodes
The notation for both MergeNodes and DecisionNodes is a diamond-shaped symbol.

15.3.4.3 Merge Nodes and Decision Nodes
A decisionInput on a DecisionNode is notated in a note symbol attached to the DecisionNode symbol, with the keyword «decisionInput», as shown in Figure 15.33.
4.3. Decision Nodes

[Diagram showing a decision node with conditions:]
- [else] [A=15]
- [no]
- [yes]

[DecisionInput]

A > B + 3
Initial Node

15.3.4.1 Initial and Final Nodes

InitialNodes are notated as a solid circle

Final Nodes

15.3.4.1 Initial and Final Nodes

ActivityFinalNodes are notated as a solid circle within a hollow circle, as shown in Figure 15.28. This can be thought of as a goal notated as “bull’s eye,” or target.
**Note:** It’s not possible to create this as one element so it’s separated to “Final Node (internal)” and “Final Node (external).

### Object Nodes

#### 15.4.4.1 Object Nodes

ObjectNodes are notated as rectangles
Example
Links

See also:

UML Activity Diagrams: Reference
9.2.4.1 Classifiers

The default notation for a Classifier is a solid-outline rectangle containing the Classifier’s name, and with compartments separated by horizontal lines below the name. The name of the Classifier should be centered in boldface. For those languages that distinguish between uppercase and lowercase characters, Classifier names should begin with an uppercase character.

```cpp
class Foo {
    /* ... */
};
```

```cpp
class A {
    public:
        int foo;
        bool boo;

    float bar();
    double baz(int val, bool cond);
};
```
10.4.4 Notation

An Interface may be designated using the default notation for Classifier (see 9.2.4) with the keyword «interface».

9.2.4.1 Classifiers

If the default notation is used for a Classifier, a keyword corresponding to the metaclass of the Classifier shall be shown in guillemets above the name.

```cpp
class Foo {
    public:
        virtual bool foo();
};
```

9.2.4.1 Classifiers

The name of an abstract Classifier is shown in italics, where permitted by the font in use. Alternatively or in addition, an abstract Classifier may be shown using the textual annotation {abstract} after or below its name.

```cpp
class A {
    public:
        virtual void boo(int x) = 0;
};
```
9.2.4.1 Classifiers

Any compartment which contains notation for Features may show those Features grouped under the literals public, private and protected, representing their visibility. The visibility literals are left-justified in the compartment with the Features’ notation appearing indented beneath them. The groups may appear in any order. Visibility grouping is optional: a conforming tool need not support it.

class A {
    public:
        int boo;

    private:
        bool foo;

    public:
        void bar();

    private:
        void baz();
};

Generalization

9.2.4.2 Other elements
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.

```cpp
class Foo { /* ... */};
class Boo { /* ... */};
class Bar: public Foo, public Boo { /* ... */};
```

![Diagram of Generalization]

**Usage**

### 7.7.4 Notation

A Dependency is shown as a dashed arrow between two model Elements. The model Element at the tail of the arrow (the client) depends on the model Element at the arrowhead (the supplier). The arrow may be labeled with an optional keyword or stereotype and an optional name (see Figure 7.18).

```cpp
class Foo {
    public:
    void foo();
};
class Boo {
    public:
    void boo(Foo& x) {
        return x.foo();
    }
};
```
Factory

class Foo { /* ... */ };  
class Boo {
    public:
    Foo* make_foo() {
        return new Foo();
    }
};

See also:
Abstract Factory Design Pattern

Realization

7.7.4 Notation
A Realization is shown as a dashed line with a triangular arrowhead at the end that corresponds to the realized Element.
class Boo: public Foo {
    public:
    virtual void foo() { /* ... */ }
};

Composition

11.5.4 Notation

Any Association may be drawn as a diamond (larger than a terminator on a line) with a solid line for each Association memberEnd connecting the diamond to the Classifier that is the end’s type.

11.5.4 Notation

An Association end is the connection between the line depicting an Association and the icon (often a box) depicting the connected Classifier. A name string may be placed near the end of the line to show the name of the Association end.

11.5.4 Notation

A binary Association may have one end with aggregation = AggregationKind::shared or aggregation = AggregationKind::composite. When one end has aggregation = AggregationKind::shared a hollow diamond is added as a terminal adornment at the end of the Association line opposite the end marked with aggregation = AggregationKind::shared. The diamond shall be noticeably smaller than the diamond notation for Associations. An Association with aggregation = AggregationKind::composite likewise has a diamond at the corresponding end, but differs in having the diamond filled in.

9.5.3 Semantics

Indicates that the Property is aggregated compositely, i.e., the composite object has responsibility for the existence and storage of the composed objects.
9.5.3 Semantics

Composite aggregation is a strong form of aggregation that requires a part object be included in at most one composite object at a time. If a composite object is deleted, all of its part instances that are objects are deleted with it.

11.5.3.1 Associations

The multiplicities at the other ends of the association determine the number of instances in each partition. So, for example, 0..1 means there is at most one instance per qualifier value.

11.5.4 Notation

An Association end is the connection between the line depicting an Association and the icon (often a box) depicting the connected Classifier. A name string may be placed near the end of the line to show the name of the Association end. The name is optional and suppressible.

```cpp
class A { /* ... */ };

class B {
    public:
        int x;
        bool y;

    A a[4];
};
```

See also:

UML Association Reference
**Aggregation**

```cpp
class B;

class A {
  public:
    A(B& b) : b_(b) {} 
  
  private:
    B& b_; 
};

class B {
  public:
    void add(A& a) {
      a_.push_back(&a); 
    }
  
  private:
    std::vector<A*> a_; 
};
```

**Note:** Tip sections with headers like 14.3.3.1 StateMachineExtension refers to UML 2.5 standard.