# UML - Class Diagrams

Which techniques do you know to describe software?

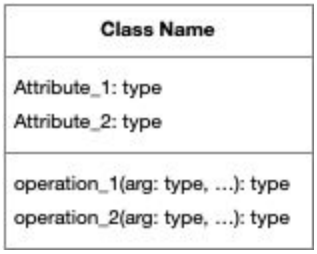
What is important when describing software?

Have you guys ever heard of UML?
- Grady Booch, Ivar Jacobson and James Rumbaugh
- 1994/95

UML has several kinds of diagrams. Do you know any of them?

A class diagram is used to describe classes. Which information should such diagram contain?
- Name
- Attributes
    - Fields
    - Methods

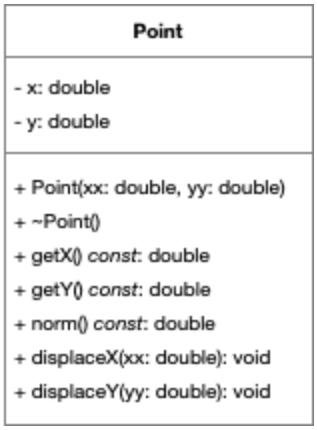
Below we can see the general format of a class Diagram in UML:




For instance, how could we model the class below?

```
class Point {
  public:
  Point(const double xx, const double yy): x(xx), y(yy) {}
  ~Point() {
      std::cout << "Object " << to_string() << " died\n";
  }
  std::string to_string() const {
      return "(" + std::to_string(x) + ", " + std::to_string(y) + ")";
  }
  double getX() const {
      return x;
  }
  double getY() const {
```

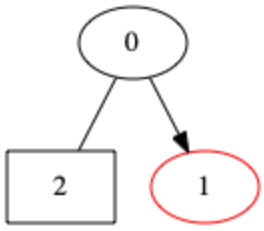
```
        return y;
    }
    double norm() const {
        return sqrt(x * x + y * y);
    }
    void displaceX(const double xx) {
        x += xx;
    }
    void displaceY(const double yy) {
        y += yy;
    }
    private:
    double x;
    double y;
};
```




Let's try to model a program that generates graphs in the dot format.

Have you ever heard of the dot format?

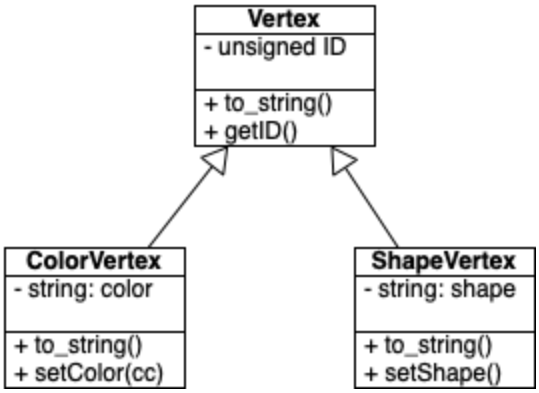
```
digraph Ex1 {
  2 [shape=box];
  1 [color=red];
  0 -> 1;
  0 -> 2 [dir=none];
}
```




We have three types of vertices:

- Simple vertices, which only have an ID
- Colored vertices, that have also a color
- Shape vertices, that have a shape (box, triangle, ellipses, etc)

How can we model these vertices?



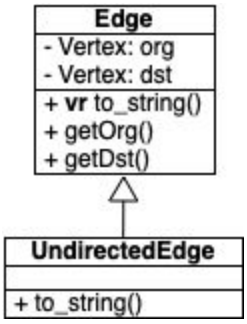

What is the meaning of the arrow with the open triangle?

Edges contain two vertices: the origin and the destination. How can we model this relation?

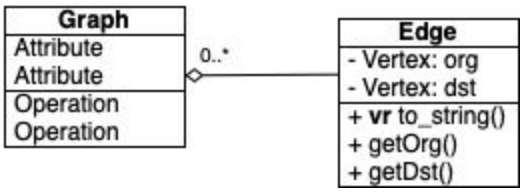

And we have two kinds of edges:

- Simple edges, which contain an arrow from origin to destination
- Undirected edges

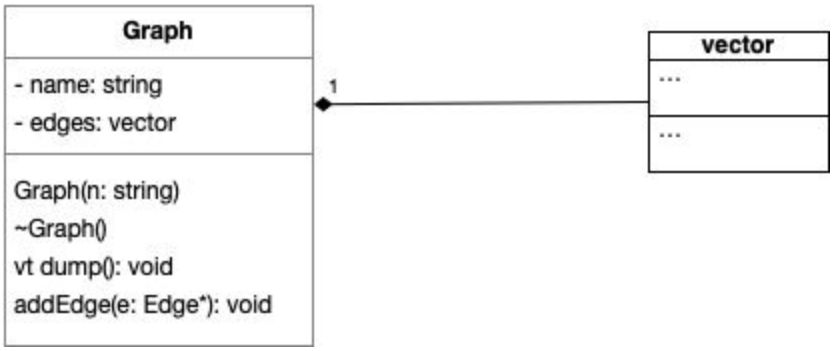
How can we model this relation?




A graph is an ensemble of edges. How can we model a graph?

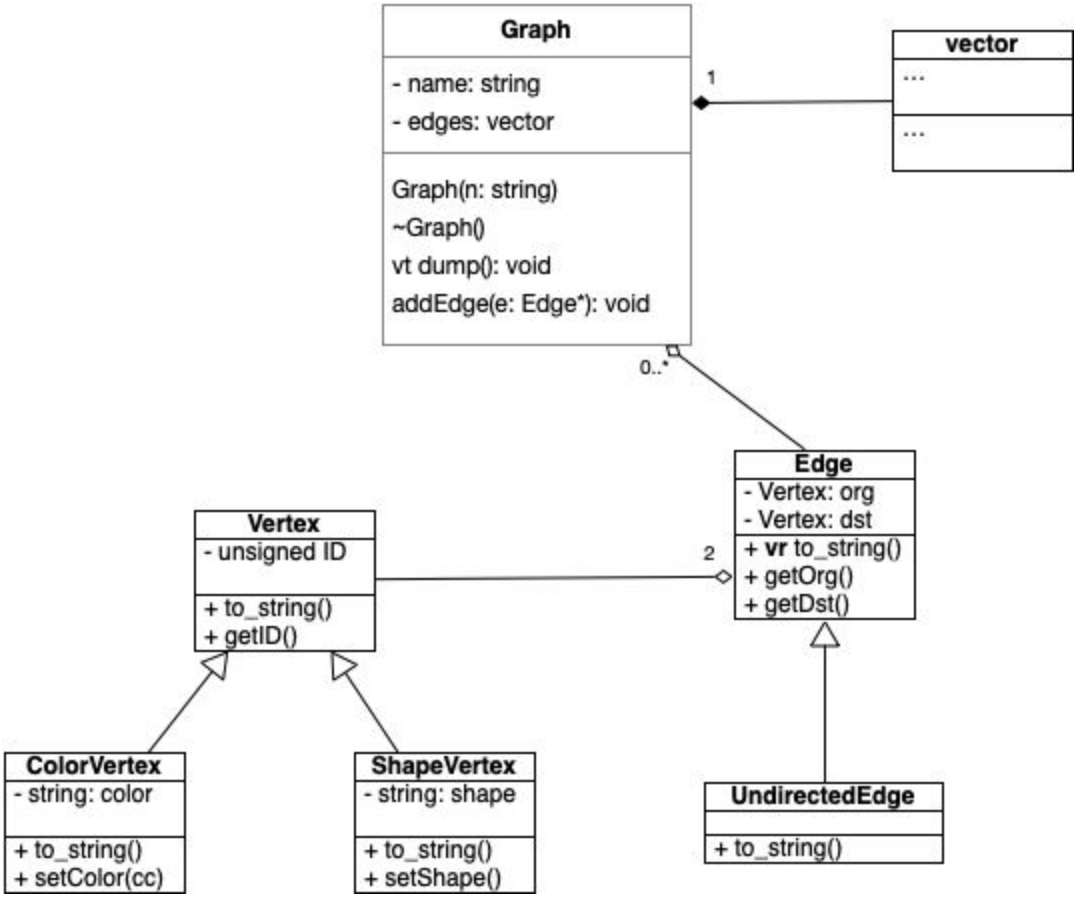



We need a data-structure to store the edges within the graph. Which data-structure?




What is the difference between the blank diamond, and the filled diamond?

Ok, how does the full model look like?




We need to implement these programs. Can you start with Vertex?

```
#ifndef VERTEX_H
#define VERTEX_H

class Vertex {
  public:
      Vertex(unsigned newId): ID(newId) {}
      virtual std::string to_string() const {
      return std::to_string(getID());
      }
      unsigned getID() const {
      return ID;
      }
  private:
      const unsigned ID;
};

#endif
```

Why do we have the #ifndef/#endif pair in this file?

Let's now move on to the subtypes of Vertex. Can you implement them?

```
#ifndef COLOR_VERTEX_H
#define COLOR_VERTEX_H

class ColorVertex: public Vertex {
  public:
      ColorVertex(const unsigned ID): Vertex(ID) {
      color = "black";
      }
      std::string to_string() const {
      return Vertex::to_string() + " [color=" + color + "]";
      }
      void setColor(std::string cc) {
      color = cc;
      }
  private:
      std::string color;
};

#endif
```

And now the ShapeVertex. Can you implement it?

```
#ifndef SHAPE_VERTEX_H
#define SHAPE_VERTEX_H

class ShapeVertex: public Vertex {
  public:
      ShapeVertex(const unsigned ID): Vertex(ID) {
      shape = "box";
      }
      std::string to_string() const {
      return Vertex::to_string() + " [shape=" + shape + "]";
      }
      void setShape(std::string ss) {
      shape = ss;
      }
  private:
      std::string shape;
};
```

```
#endif
```

Ok, What about the edges?

```
#ifndef EDGE_H
#define EDGE_H

class Vertex;

class Edge {
  public:
      Edge(const Vertex *v1, const Vertex *v2): org(v1), dst(v2) {}
      virtual std::string to_string();
      const Vertex* getOrg() const {
      return org;
      }
      const Vertex* getDst() const {
      return dst;
      }
  private:
      const Vertex *org;
      const Vertex *dst;
};

#endif
```

Oops, what the heck is that line in red doing in this program?

We need to implement Edge::to_string. That will require an implementation file. Can you produce it?

```
#include <string>

#include "Vertex.h"
#include "Edge.h"

std::string Edge::to_string() {
  return std::to_string(org->getID()) + " -> " +
std::to_string(dst->getID());
}
```

Why does it make sense to separate Edge into a header and an implementation?

Can you implement the UndirectedEdge class?

```
#ifndef UNDIRECTED_EDGE_H
#define UNDIRECTED_EDGE_H

class UndirectedEdge: public Edge {
  public:
      UndirectedEdge(const Vertex *v1, const Vertex *v2): Edge(v1,
v2) {
      }
      std::string to_string() {
      return Edge::to_string() + " [dir=none]";
      }
};

#endif
```

Now, the graph. Would you implement the header file?

```
#ifndef GRAPH_H
#define GRAPH_H

class vector;
class string;

class Graph {
  public:
      Graph(std::string nn);
      ~Graph();
      virtual void dump() const;
      void addEdge(Edge* e);
  private:
      std::string name;
      std::vector<Edge*> *edges;
};

#endif
```

And now, can you provide the implementation of this header?

```
#include <vector>
#include <iostream>
```

```
#include "Edge.h"
#include "Graph.h"

Graph::Graph(std::string nn): name(nn) {
  edges = new std::vector<Edge*>();
}

Graph::~Graph() {
  delete edges;
}

void Graph::addEdge(Edge* e) {
  edges->push_back(e);
}

void Graph::dump() const {
  std::cout << "digraph " << name << " {\n";
  for (Edge *e: *edges) {
      std::cout << "  " << e->to_string() << ";\n";
  }
  std::cout << "}\n";
}
```

And we need a driver to test these programs. Let's implement it:

```
#include <string>
#include <iostream>

#include "Vertex.h"
#include "ColorVertex.h"
#include "ShapeVertex.h"
#include "Edge.h"
#include "UndirectedEdge.h"
#include "Graph.h"

int main() {
  Vertex v0(0);
  ColorVertex v1(1);
  ShapeVertex v2(2);
  v1.setColor("red");
  Edge e0(&v0, &v1);
  UndirectedEdge e1(&v0, &v2);
  Graph g0("Ex1");
```

```
  g0.addEdge(&e0);
  g0.addEdge(&e1);
  g0.dump();
}
```

Hum… I would like to print the attributes of the vertices, but they are not there. See what we get:

```
digraph Ex1 {
  0 -> 1;
  0 -> 2 [dir=none];
}
```

However, we would like to get something like this code below:

```
digraph Ex1 {
  2 [shape=box];
  1 [color=red];
  0;
  0 -> 1;
  0 -> 2 [dir=none];
}
```

Can you modify Graph.cpp to account for these attributes?

```
#include <set>
#include <vector>
#include <iostream>
#include "Edge.h"
#include "Vertex.h"
#include "Graph.h"

Graph::Graph(std::string nn): name(nn) {
  edges = new std::vector<Edge*>();
}

Graph::~Graph() {
  delete edges;
}

void Graph::addEdge(Edge* e) {
  edges->push_back(e);
}
```

```
void Graph::dump() const {
  std::cout << "digraph " << name << " {\n";
  // Print the vertices:
  std::set<const Vertex*> vertices;
  for (Edge *e: *edges) {
     vertices.insert(e->getOrg());
     vertices.insert(e->getDst());
  }
  for (const Vertex *v: vertices) {
     std::cout << "  " << v->to_string() << ";\n";
  }
  // Print the edges:
  for (Edge *e: *edges) {
     std::cout << "  " << e->to_string() << ";\n";
  }
  std::cout << "}\n";
}
```

But now the model is a bit different. How the actual model would like like now?