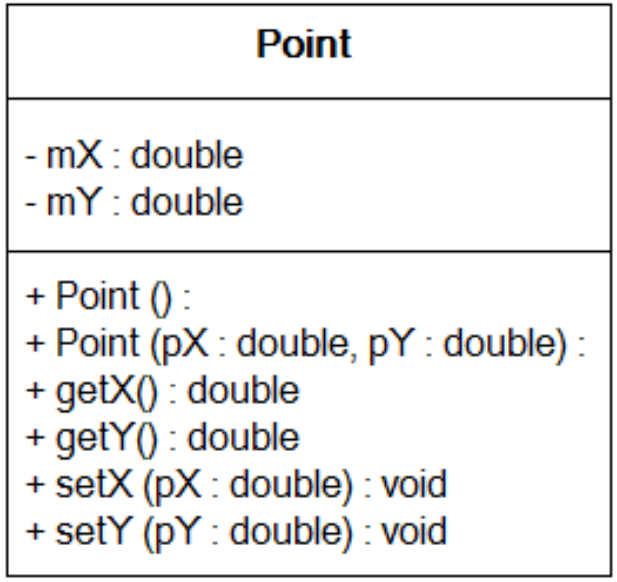
1. Object Oriented Design :: Introduction to UML and Class Diagrams

UML (Unified Modeling Language) is a standard language for documenting the design of a computer program or computer system.

Designing a large program is every bit as complex as designing a skyscraper or an advanced airliner. In this section we will discuss a small part of UML, with a focus on **UML class diagrams**.

A **UML class diagram** is a diagram that describes the structure of a system by showing the system's classes, their data members (attributes), methods (behaviors), and the relationships among classes and objects. For example, here is the UML diagram for the *Point* class.

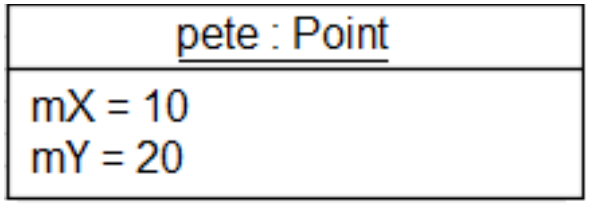


1. Object Oriented Design :: UML Object Diagrams

A **UML object diagram** is a diagram that shows a complete or partial view of an object at a specific time during program execution. For example, suppose we declare and instantiate a *Point* object named *pete*:

```
Point pete = new Point(10, 20);
```

We can represent *pete* by an object diagram:



In an object diagram, the top box contains the name of the object followed by a colon followed by the class of an object; all of this text is underlined. The bottom box contains the relevant data members and their values at a this moment in time.

1. Object Oriented Design :: Class Dependency Relationship

In any OO design, there will be relationships among and between classes. The first relationship we will discuss is a **dependency relationship**.

We say class C **depends** on class D if class D is used to **declare a parameter** or **local variable of a method** in class C. For example, suppose we have a *Canvas* class that is used to draw on a graphical display and one of the methods of *Canvas* is *drawPoint*():

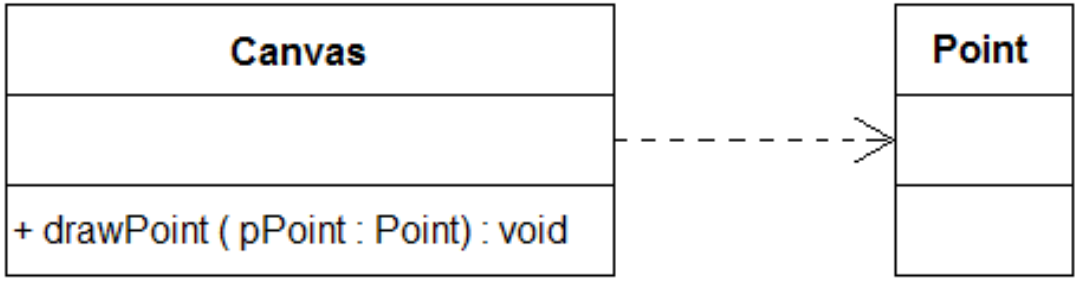
```
public class Canvas {
  ...
  public void drawPoint(Point pPoint) {
    ...
  }
}
```

Canvas is the **dependent class** because it is **dependent** on class *Point*. If the *Point* class were to change, this dependency relationship tells us the *Canvas* may also need to be modified.

1. Object Oriented Design :: Class Dependency Relationship (continued)

However, *Point* does not use the *Canvas* class so *Point* is not dependent on *Canvas* (*Point* is **independent** of *Canvas*) and we refer to *Point* as an **independent class** in this relationship.

A **dependency relationship** is documented in the UML design by a dashed line with an open arrow pointing from the **dependent class** to the **independent class**:



A dependency relationship forms a "knows about" relationship. *Canvas* knows about *Point* (i.e., that the *Point* class exists) but *Point* does not know about *Canvas* (*Point* does not know that the *Canvas* class exists).