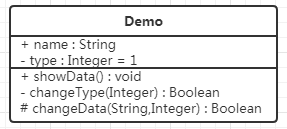
**UML基础**

###### 类的UML图



Java代码：

|  |
| --- |
| class Demo {  public String name;  private Integer type = 1;    public void showData() {}  private Boolean changeType(Integer type) {  return true;  }  protected Boolean changeData(String name, Integer type) {  return true;  }  } |

说明：

类图结构：

第一行为类名，接口加上 <<interface>>

第二行为属性，[]中为可选值

+/-/# 属性名 : 类型 [ = 默认值]

第三行为方法

+/-/# 方法名(参数) : 返回类型

符号说明：

+ : public

- : private

# : protected

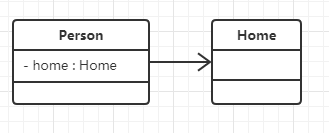
静态属性、方法，加上下划线如 + getInstance() : Object

###### 类之间的关系

* 1. 关联关系

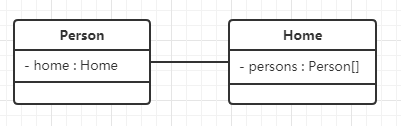
类之间的结构化关系

* + 1. 单向关联



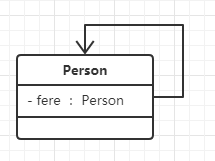
Person对象拥有一个Home属性

* + 1. 双向关联



Person对象拥有一个Home属性，Home又拥有多个Person属性

* + 1. 自关联



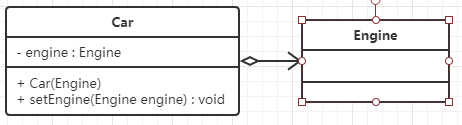
Person对象拥有一个fere(伴侣)属性，同事fere也是一个Person对象

* + 1. 多重性关联

关联的两个对象在数量上的关系

* + 1. 聚合关系

成员对象是整体对象的一部分，但是成员对象可以脱离整体对象独立存在。在实例化时，可以在构造器传参赋值，或者通过方法(setter)来赋值。

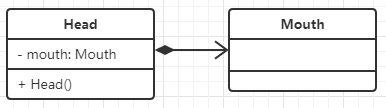


发动机(Engine)是汽车(Car)的组成部分，但是发动机可以独立存在用于别的地方如船只，因此，汽车和发动机是聚合关系。

|  |
| --- |
| class Car {  private Engine engine;    public Car(Engine engine) {  this.engine = engine;  }  public void setEngine(Engine engine) {  this.engine = engine;  }  }  class Engine {} |

* + 1. 组合关系

也表示整体和部分的关系，但是在组合中，一旦整体对象不存在那么局部的那个对象也将不存在。在实例化时，一般是在整体实例化时才实例化局部。

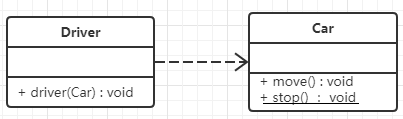


人的头(Head)与嘴巴(Mouth)，嘴巴是头的组成部分之一，而且如果头没了，嘴巴也就没了，因此头和嘴巴是组合关系。

|  |
| --- |
| class Head {  private Mouth mouth;  public Head() {  mouth = new Mouth();  }  }  class Mouth {} |

* 1. 依赖关系

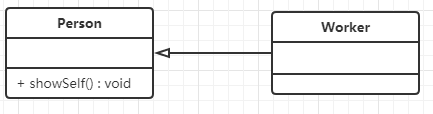
表示某个对象(Driver)需要另一个对象(Car)才能使用。实现方式包括(见代码)，作为参数传递、作为局部变量、使用静态方法。



在Driver类的drive()方法中将Car类型的对象car作为一个参数传递，以便在drive()方法中能够调用car的move()方法，且驾驶员的drive()方法依赖车的move()方法，因此类Driver依赖类Car。

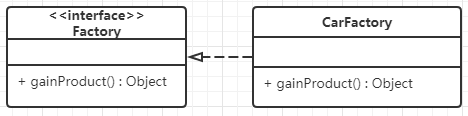
|  |
| --- |
| class Driver {  public void driver(Car car) {  //作为参数传递  car.move();  //通过作为局部变量  Car smart = new Car();  smart.move();  //使用静态方法  Car.stop();  }  }  class Car {  public void move() {};  public static void stop() {};  } |

* 1. 泛化关系(继承关系)



|  |
| --- |
| class Person {  public void showSelf() {  System.out.println("i am a person.");  }  }  class Worker extends Person {  @Override  public void showSelf() {  super.showSelf();  System.out.println("i am a worker.");  }  } |

* 1. 接口与实现



|  |
| --- |
| interface Factory {  public Object gainProduct();  }  class CarFactory implements Factory {  @Override  public Object gainProduct() {  System.out.println("yield a car.");  return new Car();  }  } |