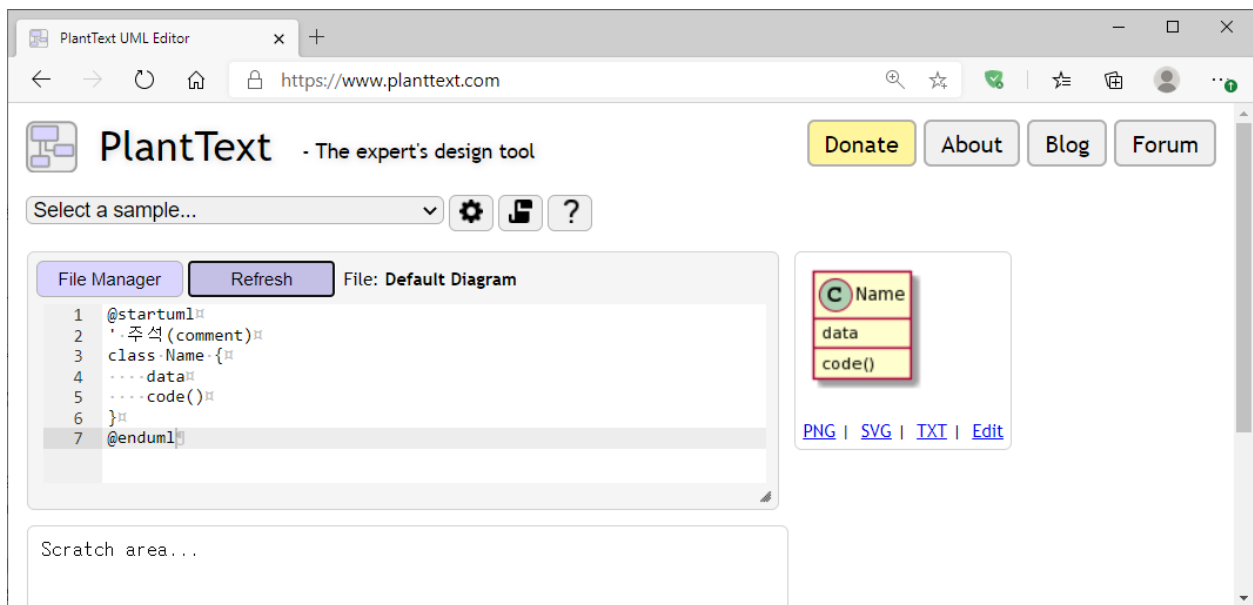
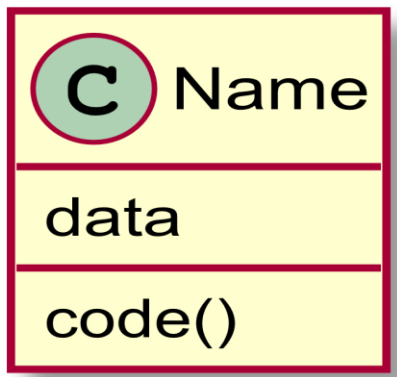


부록 4. UML 로 클래스 그려보기

다음 그림은 책에서 보인 UML 형태로 표현한 클래스 다이어그램이다. 아주 단순한 형태로 그렸는데, 이 그림은 간단한 스크립트를 이용해서 UML 을 그려주는 PlantUML(<http://plantuml.com>)을 사용한다. PlantUML 은 다양한 인텔리제이 IDEA 나 다양한 프로그래머 에디터와 통합될 수 있어 그런 것들을 사용해도 된다. 여기서는 어디서나 사용할 수 있도록 PlantUML 을 웹에 구현한 <http://planttext.com> 사이트를 이용해서 그림을 그린다.



앞에서 보인 그림은 planttext.com 에서 위에서 보인 그림을 만들어내는 화면을 보인 것이다. 그림에서 보인 것처럼 왼쪽에는 텍스트를 입력할 수 있는 텍스트 에디터 같은 것이 있고 오른쪽에는 클래스 다이어그램이 보인다. 그리고자 하는 클래스 다이어그램을 왼쪽에 있는 에디터에 입력하고 “Refresh”버튼을 클릭하면, 오른쪽에 바뀐 그림이 나타난다. 필요하면 원하는 형태로 저장하는 것도 가능하다. 그럼 planttext.com 에 접속해서 다음 코드를 입력해보고, 앞에서 보였던 실제 클래스 다이어그램을 만들어보도록 한다.

소스코드 A3.1

```
@startuml
' 주석(comment)
class Name {
    data
    code()
}
@enduml
```

PlantUML 코드는 소스코드 A3.1 에서 보인 것처럼 “@startuml”로 시작하고 “@enduml”로 마무리된다. 그 사이에 UML 을 표현하는데 필요한 코드를 넣으면 된다. 일반 코드를 작성하는 것처럼 주석문을 넣을 수도 있고 클래스 다이어그램을 구성하는 코드를 넣을 수도 있다. 시퀀스 다이어그램 등을 그릴 수 있는 기능도 있지만, 여기서는 클래스 다이어그램을 그리는 데 필요한 내용만을 다루기로 한다. 다음은 PlantUML 의 간단한 사용법을 보인다.

PlantUML 에서 주석 표시하기

참고로 PlantUML 에서 작은 따옴표(') 한 개로 시작하는 문장은 코드의 주석처럼 취급된다.

소스코드 A3.2

```
@startuml
'Doctor, Patient, HospitalAppointment 는 앞에서 작성한 내용과 같음
class Doctor
class Patient
class HospitalAppointment

HospitalAppointment --> Doctor
HospitalAppointment --> Patient
@enduml
```

PlantUML 에서 클래스 표현하기

PlantUML 에서 클래스를 표현하는 것은 앞에서 보인 것처럼

```
class 이름 {
}
```

또는

```
class 이름
```

형태로 작성한다. 첫 번째 방법은 클래스 내부에 멤버 변수와 함수 등을 넣을 때 주로 사용하고 두 번째는 단순히 클래스의 상관 관계만을 표시하거나 클래스의 내부 구성과는 관계없이 클래스의 상징성이 필요할 때 사용한다. UML 에서 클래스의 속성이나 함수를 표기하는 방법은 다음과 같다.

UML 에서 객체에 속성과 메소드 추가하기

UML 클래스 다이어그램에 속성을 추가하는 표기 방법은 왼쪽에 속성의 이름을 적고 콜론(:)을 쓴 다음에 오른쪽에 자료형을 적는다. 즉 **** 속성_이름 : 자료형**** 형태로 적는다. 만약 초기값을 지정해야 한다면

속성_이름 : 자료형 = 초기값 형태로 작성한다. 함수를 표현하는 방법은

함수_이름([매개_변수_리스트]) : 반환하는 자료형으로 적는다. [매개_변수_리스트]는 "속성_이름 : 자료형" 형태로 표기되며 매개 변수가 없다면 생략할 수 있다.

자료형은 여기서는 문자열을 나타내는 String 형을 사용한다. 다음에서 [] 부분은 생략 가능하다.

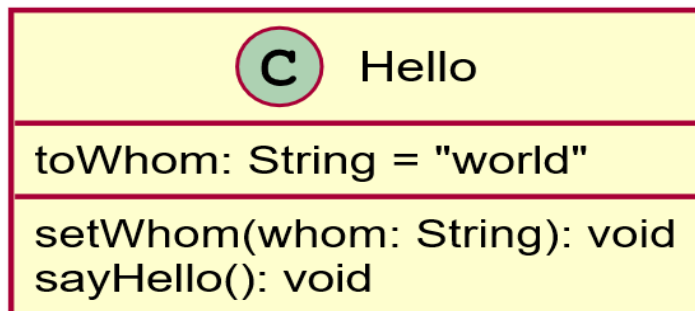
```
속성_이름 : 자료형 [= 초기값]
함수_이름([매개_변수_리스트]) : 반환하는_자료형
```

여기서는 앞에서 만들었던 Hello 클래스에 인사하는 대상의 이름을 넣어 본다.

```
@startuml
class Hello {
    toWhom : String = "world"
    sayHello() : void
}
@enduml
```





아래는 위에서 사용된 속성에 대한 설명과 UML 다이어그램을 보인다.

데이터속성에 사용된 요소	설명
toWhom	속성명으로 첫글자는 소문자, 그 다음에 나오는 단어의 시작은 대문자로 의미있게 지음
:	속성명과 자료형을 분리하는 목적으로 사용됨
String	자료형
"world"	값을 생성 시점에 초기화시킴. 즉 초기값



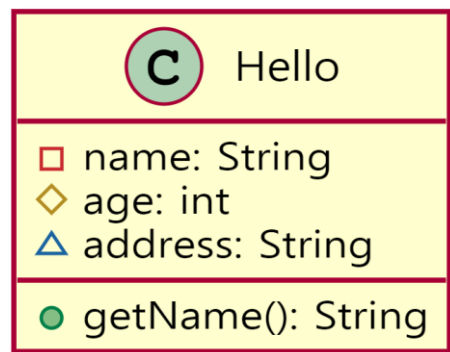
접근제어자 표기

다음 표는 접근 제어자를 UML, PlantUML, 클래스 다이어그램에서 표현하는 것을 보인다. 표에서 볼 수 있듯이 PlantUML 에서 접근 제어자를 표기하는 것은 UML 에서 표현하는 것과 동일하다. 예를 들어 멤버 변수나 함수 앞에 '+'를 붙이면 public 접근 제어자를 나타낸다. 표의 가장 오른쪽 열은 planttext.com 에서 만들어지는 클래스 다이어그램에서 접근제어자를 표현하는 것을 보인다.

구분	UML 표현 방법	PlantUML 표현 방법	클래스 다이어그램 표현
public	+	+	
protected	#	#	
default (package private)	~	~	
private	-	-	

다음은 클래스를 한 개 만들고 다른 접근 제어자를 멤버 변수와 함수에 지정한 코드와 이를 이용해서 만든 클래스 다이어그램의 예를 보인다.

```
@startuml
class Hello {
    -name: String
    #age: int
    ~address: String
    +getName(): String
}
@enduml
```



PlantUML 에서 관계 작성하기

여기서는 UML 클래스 다이어그램에서 클래스 사이의 상관 관계를 표현하는 방법을 간략하게 살펴본다. UML 에 대해서 자세히 배우는 것이 이 책의 목표는 아니므로, 여기서는 간단하게 설명하고, 좀 더 자세한 내용은 UML 관련 책이나 인터넷 자료를 찾아보도록 한다.

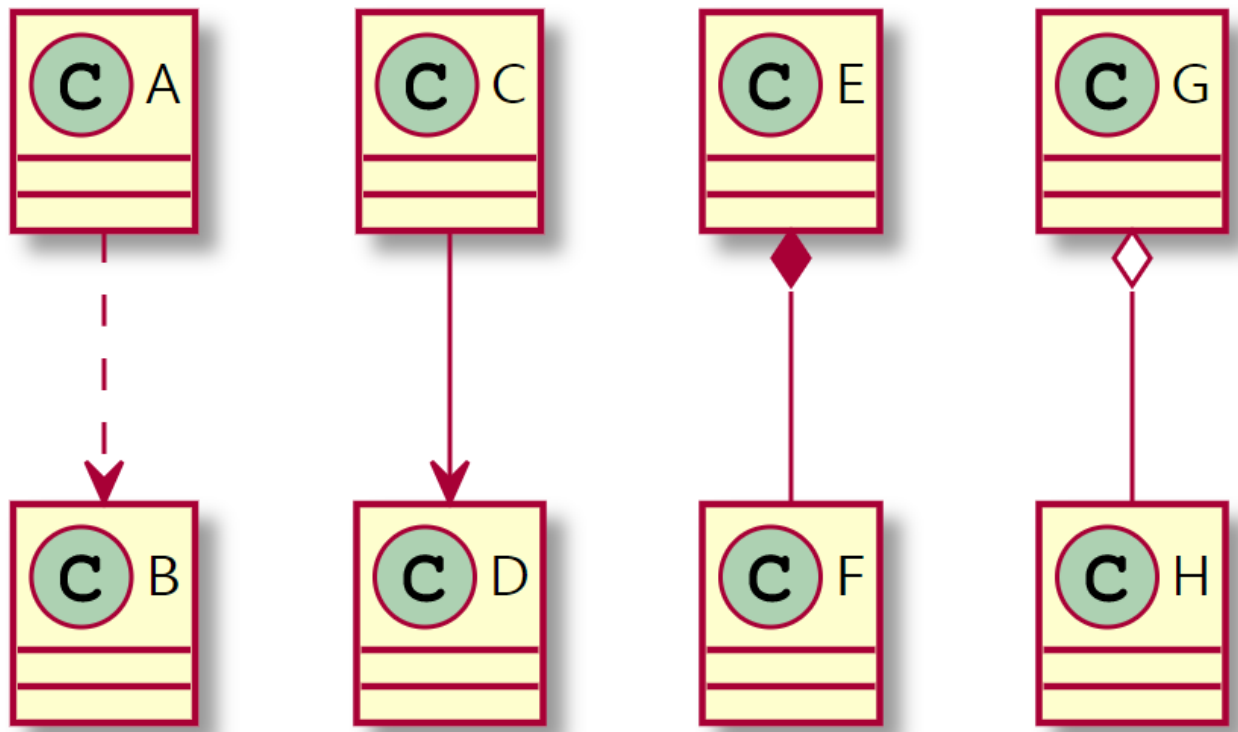
PlantUML 에서 연관, 의존, 집합, 구성을 표현하는 방법은 다음과 같다.

관계	PlantUML 스크립트 작성 방법
연관	-- (실선, 양방향), --> (실선 화살표, 단방향)
의존	.. (점선, 양방향), ..> (점선 화살표, 단방향)
집합	o-
구성	*-

PlantUML에서는 클래스 다이어그램에서 클래스의 상대적인 위치를 오른쪽, 왼쪽, 위쪽, 아래쪽에 둘 수 있도록 right, left, up, down 으로 지정할 수 있다. 다음은 PlantUML 로 그린 클래스 관계의 예를 몇 가지 보인다. A 와 B 는 연관 관계인데 A 에 B 의 참조를 가지고 있다. 그런데 B 를 A 의 왼쪽에 점선으로 연결하기 위해 ..>에 left 를 넣었다. C 와 D 는 C 가 전체, D 가 일부인 구성 관계이다. E 와 F 는 구성을 나타내며 E 가 전체, F 가 부분을 나타낸다. 단 E 와 F 는 집합 관계를 보인다.

소스코드 A3.3

```
@startuml
class A
class B
class C
class D
class E
class F
class G
class H
C --> D
A ..> B
E *-- F
G o-- H
@enduml
```



PlantUML 에서 상속 작성하기

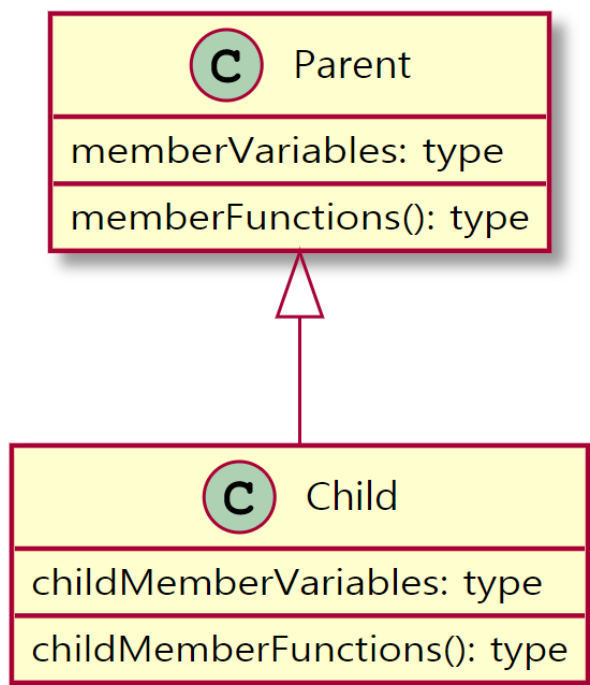
PlantUML 에서 상속을 표현하는 방법은 두 가지가 있다. 첫 번째는 클래스 코드를 작성하면서 자바 코드처럼 `extends` 를 이용해서 표현하는 것이다. 예를 들어 다음 코드와 그림은 Child 클래스가 Parent 클래스로부터 상속 받음을 보인다.

소스코드 A3.4

```
@startuml
class Parent {
    memberVariables: type
    memberFunctions(): type
}

class Child extends Parent {
    childMemberVariables: type
    childMemberFunctions(): type
}
@enduml
```

자바 코드에서 클래스 상속을 표현하는 것처럼 “`extends`”라는 키워드를 사용해서 Child 클래스를 Parent 로부터 상속 받았다. 이렇게 “`extends`”를 이용해서 상속 관계를 표현하면, 다음 그림에 보인 것처럼 빈 삼각형 화살표를 Child 에서 Parent 로 향하도록 그려진 클래스 다이어그램을 만들 수 있다.



두 번째 방법은 클래스들을 독립적으로 구성한 후에 자식으로부터 부모로 빈 삼각형 화살표를 그리기 위해 “자식 클래스 이름”과 “부모 클래스 이름” 사이에 상속 관계를 표시하는 “`--|>`”를 넣는 것이다. 예를 들어 다음 PlantUML 코드는 앞에서 보인 상속 관계를 똑같이 나타낸다.

소스코드 A3.5

```
@startuml
class Parent {
    memberVariables: type
    memberFunctions(): type
}

class Child {
    childMemberVariables: type
    childMemberFunctions(): type
}

Child --|> Parent
@enduml
```

PlantUML 에서 인터페이스와 구현 관계 작성하기

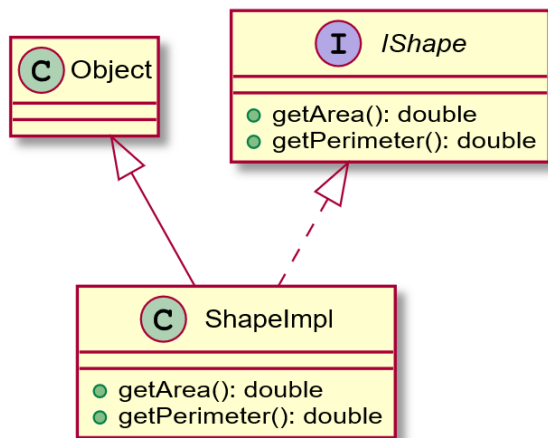
PlantUML 에서 인터페이스를 표현하려면 "class"대신 "interface"를 사용한다. 또 인터페이스를 구현하는 클래스를 표현하려면 자바 코드에서 하듯이 "implements"로 표시하거나 구현 관계 화살표를 나타내는 “..|>”를 이용한다. 다음 코드는 인터페이스와 이를 구현하는 클래스를 PlantUML 로 작성하고 이를 클래스 다이어그램을 이용해서 보인 것이다.

소스코드 A3.6

```
@startuml
class Object

interface IShape {
    +getArea(): double
    +getPerimeter(): double
}

class ShapeImpl extends Object implements IShape {
    +getArea(): double
    +getPerimeter(): double
}
@enduml
```



그림에서 볼 수 있듯이 ShapeImpl 클래스는 Object로부터 상속 받고 IShape 을 구현한 것을 보인다. UML 에서는 구현 관계를 빈 삼각형 점선 화살표를 이용해서 표현한다. 구현 관계는 상속 관계와 비슷하게 “..|>” 를 이용해서 표현할 수도 있다. 다음 PlantUML 코드도 똑 같은 다이어그램을 생성한다.

소스코드 A3.7

```

@startuml
class Object

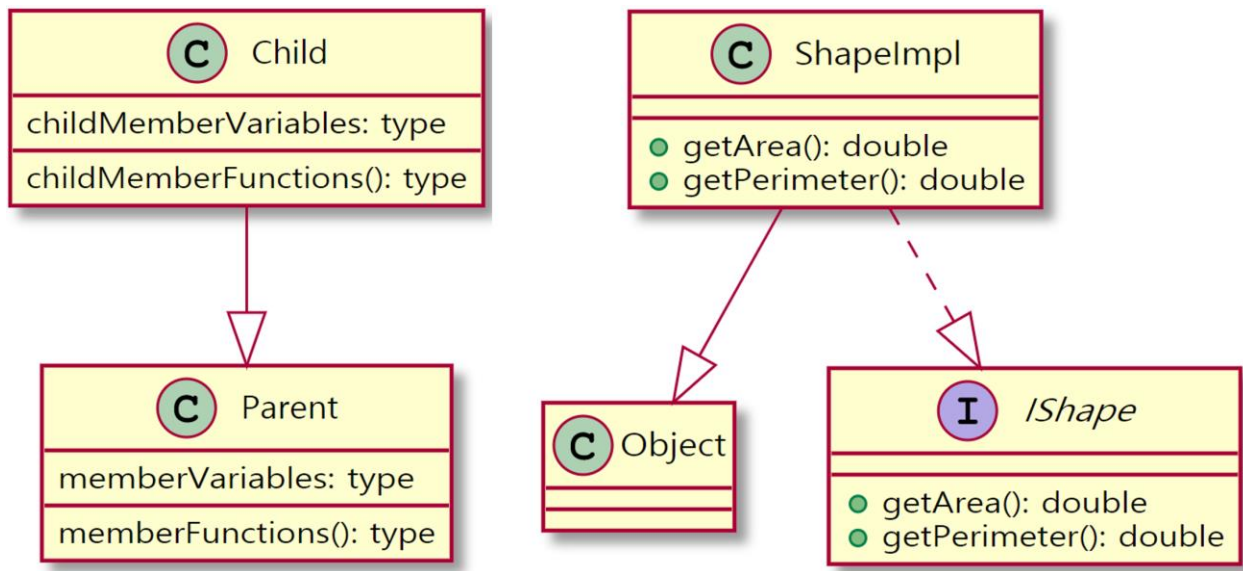
interface IShape {
    +getArea(): double
    +getPerimeter(): double
}

class ShapeImpl {
    +getArea(): double
    +getPerimeter(): double
}

ShapeImpl --|> Object
ShapeImpl ..|> IShape
@enduml
  
```

클래스들의 위치 지정하기

앞에서 “extends”나 “implements”를 이용해서 클래스간 상속 또는 구현 관계를 보이는 것과 “--|>”나 “..|>”를 이용해서 클래스 다이어그램을 구성하는 것은 동일한 결과를 보인다고 했다. 즉 부모-자식의 상속 관계나 인터페이스와 구현(구상) 클래스(concrete class)간의 관계를 보이는 것은 동일하게 보인다. 다만 PlantUML 을 그리는 과정은 프로그램에 의해서 자동으로 그려지다 보니까 우리가 원하는 다이어그램이 만들어지지 않는 경우가 있다. 예를 들어 다음 클래스 다이어그램들은 소스코드 A3.5 와 A3.7 을 plantuml.com 에서 그린 것이다.



그런데 그림을 보면 앞에서 보았던 다이어그램과는 다르게 부모-자식 클래스들의 위치가 뒤집혀 있는 것을 볼 수 있다. 화살표 방향들을 살펴보면 상속 관계 또는 구현 관계가 틀린 것은 아니고, PlantUML 이 프로그램에 의해서 자동으로 그림이 만들어지다 보니, 지금처럼 클래스들의 위치가 우리가 원하는 곳에 위치하지 않는 경우가 있을 수 있다. 이럴 때 완벽하지는 않지만 클래스들의 상대적인 위치를 지정할 수 있는 방법이 있다. 단 이 방법은 “-->”, “..>”, “--|>”, “..|>” 등을 이용해서 클래스의 상관 관계를 표현할 때만 사용할 수 있는데, “-방향-” 또는 “.방향.” 형태로 사용한다. 이 때 방향은 “right”, “left”, “up”, “down”을 사용할 수 있다. 예를 들어 소스 코드 A3.7 에서 ShapeImpl 과 Object 의 상관 관계를 표현하면서 Object 는 ShapeImpl 보다 반드시 위에 있어야 하고 IShape 은 ShapeImpl 의 오른쪽에 위치해야 한다면 다음처럼 코드를 작성할 수 있다.

```

ShapeImpl -up-|> Object
ShapeImpl .right.|> IShape

```

그럼 다시 코드를 작성하고 클래스 다이어그램을 그려본다. 앞에서 설명한 것처럼 Object 는 ShapeImpl 의 위에 그리고 IShape 은 ShapeImpl 의 오른쪽에 위치시킨다.

소스코드 A3.7

```

@startuml
class Object

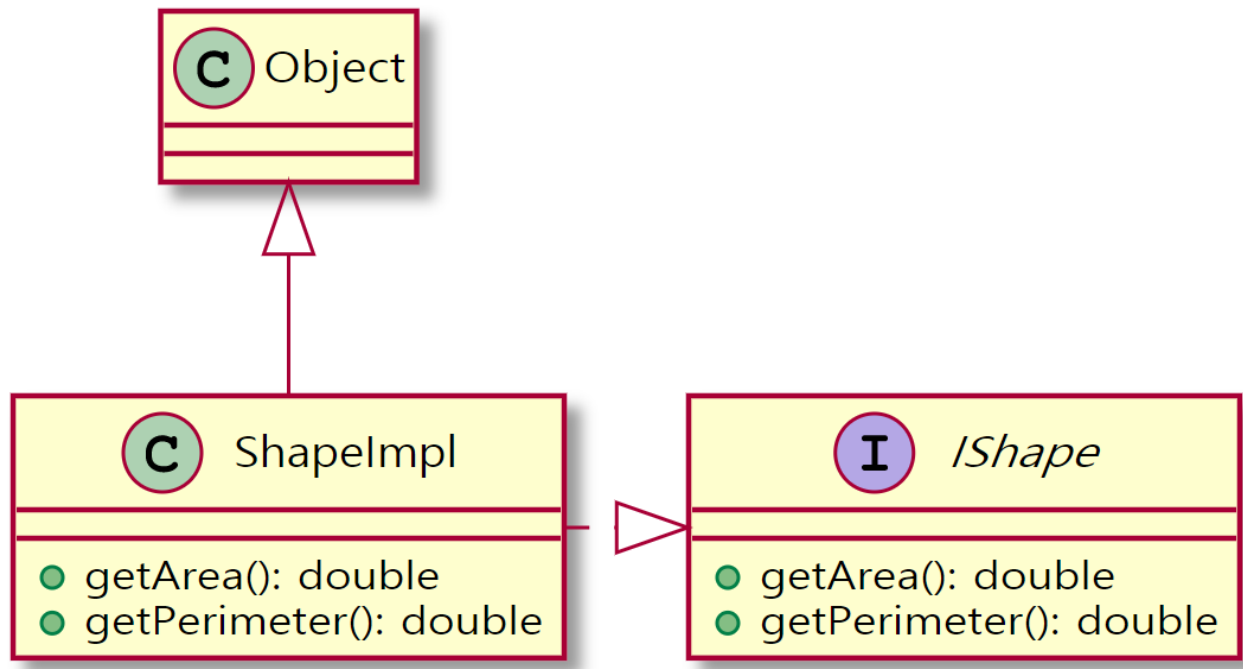
interface IShape {
    +getArea(): double
    +getPerimeter(): double
}

class ShapeImpl {
    +getArea(): double
    +getPerimeter(): double
}

ShapeImpl -up-|> Object
ShapeImpl .right.|> IShape
@enduml

```

다음은 이 코드를 이용해서 그려진 클래스 다이어그램을 보인다. 예상했던 형태의 그림이 나타남을 볼 수 있다.



클래스 관계가 이렇게 복잡하지 않은 경우에는 원하는 형태의 그림을 만들 수 있지만, 아주 복잡해지면 단순히 이렇게 오른쪽, 왼쪽, 위, 아래 정도만을 지정해서 원하는 형태의 그림을 만드는 것은 쉽지 않다. 하지만 이렇게 방향을 지정해서 어느 정도 원하는 형태의 그림을 그릴 수 있다는 정도로 알아둔다.

패키지를 UML 로 표기하기

아래 그림들은 패키지를 만들지 않은 상태의 클래스 다이어그램과 패키지로 묶여 있는 클래스들의 다이어그램을 보인다. 패키지는 웹 브라우저의 탭 모양에 클래스를 넣어서 표현하고, PlantUML 에서는 다음 형태로 표현한다. 즉 패키지에 포함되는 클래스 나열하면 된다.

```
package 패키지이름 {
    패키지에 포함되는 클래스 코드
}
```

다음은 패키지로 묶이지 않은 클래스들과 패키지에 포함된 클래스를 작성해본 것이다.

```
소스코드 A3.8
@startuml
class Rectangle {
    length: int
    width: int
    area(): int
}

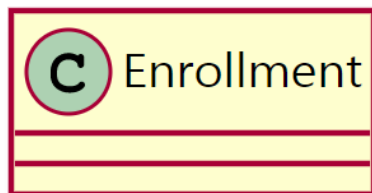
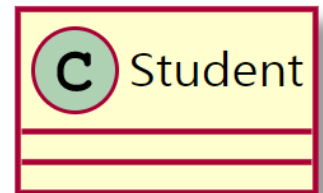
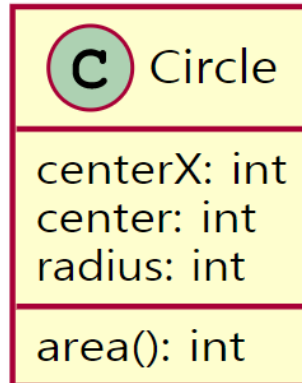
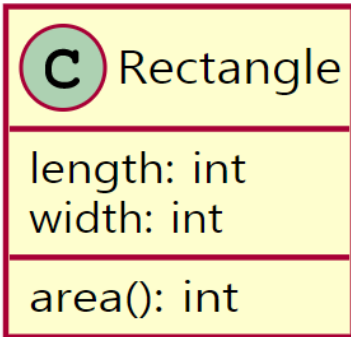
class Circle {
    centerX: int
```

```

        center: int
        radius: int
        area(): int
    }

class Student
class Enrollment
class Course
@enduml

```



클래스들이 독립적으로 구성됨을 확인할 수 있다. 다음 코드는 이 클래스들을 두 개의 패키지로 묶어서 배치하는 것을 보인다.

소스코드 A3.9

```

@startuml
package my.figures {
    class Rectangle {
        length: int
        width: int
        area(): int
    }

    class Circle {
        centerX: int
        center: int
        radius: int
    }
}

```

```

        area(): int
    }
}

package my.register {
    class Student
    class Enrollment
    class Course
}
@enduml

```

다음은 클래스 다이어그램을 보인다. my.register 와 my.figures 라는 패키지 안에 클래스들이 위치한 것을 볼 수 있다.

