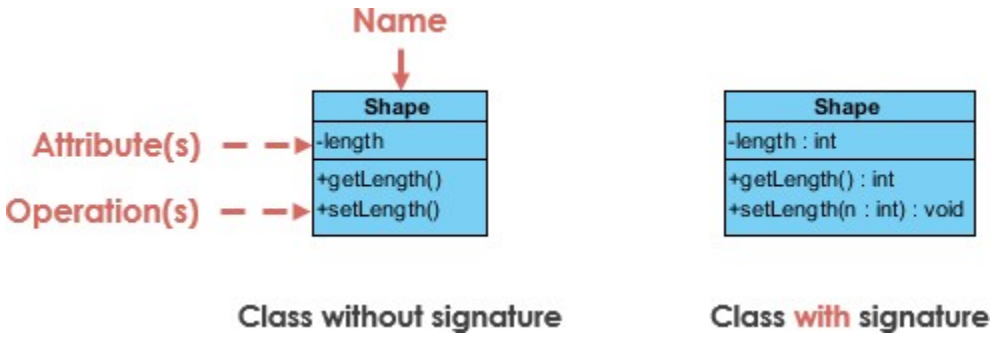
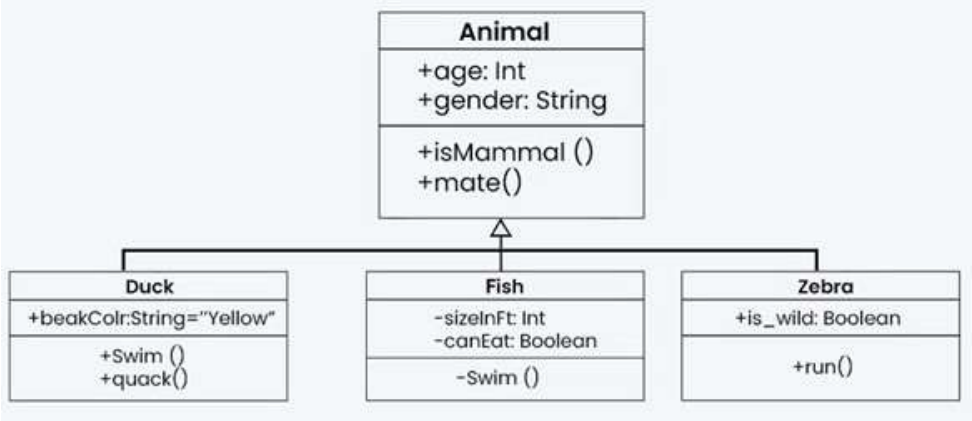
## Class Diagram

- Box with **3 sections**
- The **top contains the class name**
- The **middle lists the classes attributes**
- The **bottom lists the classes methods**
- Can indicate parameters and return types to methods, as well as their visibility
- Class diagrams also help us identify relationships between different classes or objects.

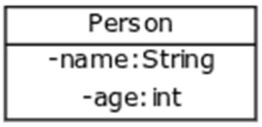



## Describing class and class attributes

First, we will describe one class and its attributes. Below is the source code for a **class called Person** which has **two class attributes name and age**.

```
public class Person {
   private String name;
   private int age; }
```




In a class diagram, class attributes should be shown as:

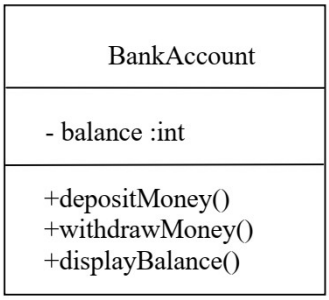
**visibility name: type multiplicity**

**Where visibility is one of:**

| Symbol | Visibility Type | Description |
|---|---|---|
| + | **public** | **Accessible from anywhere (all classes).** |
| - | **private** | **Accessible only within the owning class.** |
| # | **protected** | **Accessible within the class and its subclasses** |
| ~ | **package** | **Accessible only within the same package or namespace.** |

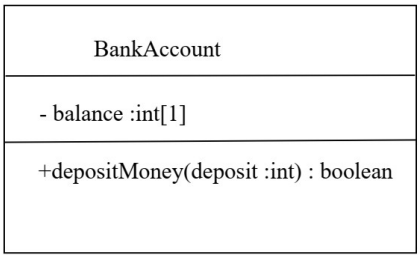
## Example

Draw a diagram to represent a **class called BankAccount** with the **attribute private balance (type int)** and public **methods depositMoney()** and **withdrawMoney()**. **Show appropriate visibility modifiers**.




## Example

Draw a diagram to represent a **class called BankAccount** with a **private attribute balance (this being a single integer)** and a **public method depositMoney()** which **takes an integer parameter, 'deposit' and returns a boolean value**. Fully specify all this information on a UML class diagram.




## Describing class constructor

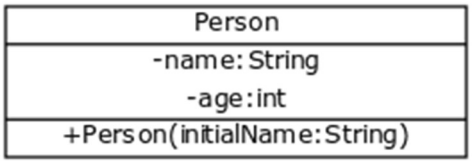
Below we have the source code for a constructor for our Person class. The constructor gets the name of the person as a parameter.

```
public class Person {
   private String name;
   private int age;
   // constructor
   public Person(String initialName) {
      this.name = initialName;
      this.age = 0;  }}
```

In a class diagram, we list the **constructor (and all other methods)** below the attributes. A line below the attributes list separates it from the method list. Methods are written with +/- (depending on the visibility of the method), method name, parameters, and their types. The constructor above is written

**+Person(initialName:String)**

The parameters are written the same way class attributes are — "parameterName: parameterType".

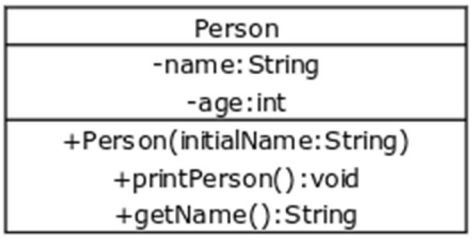



## Describing class methods

Below we have added a **method printPerson() which returns void to the Person class**.

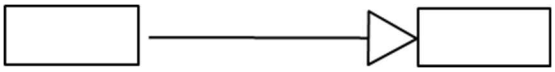
```
public class Person {
   private String name;
   private int age;
```

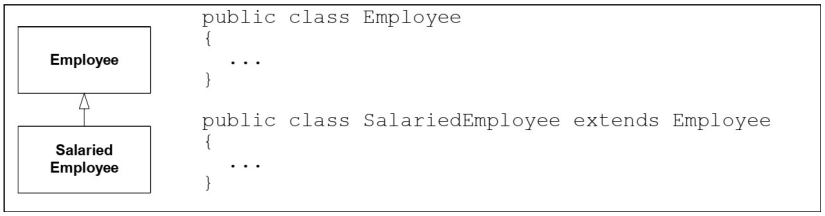
```
// constructor
public Person(String initialName) {
   this.name = initialName;
   this.age = 0;  }
public void printPerson() {
   System.out.println(this.name + ", age " +   this.age + " years");  }
public String getName() {
   return this.name;  }}
```



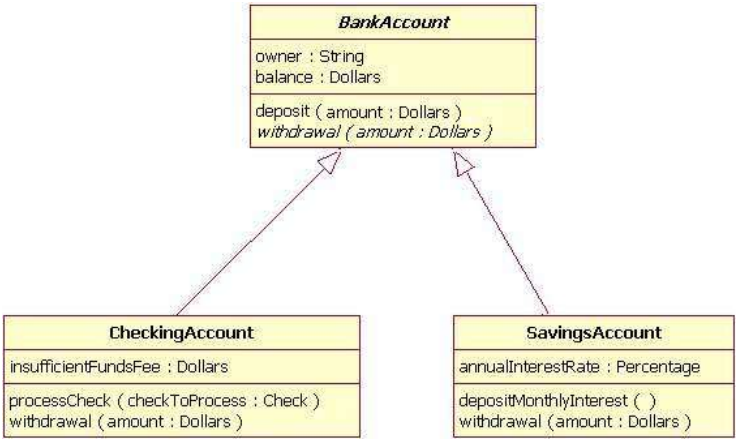

## Inheritance

Represented by a solid line with an arrow pointing from the derived class to the base class.

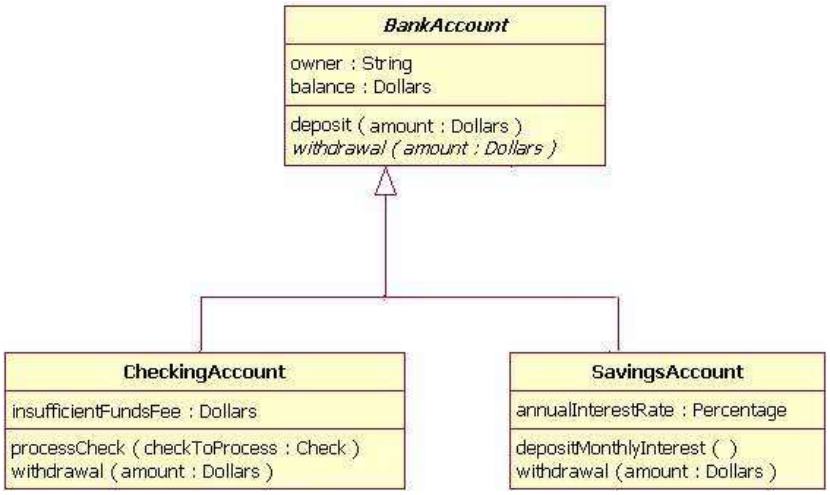




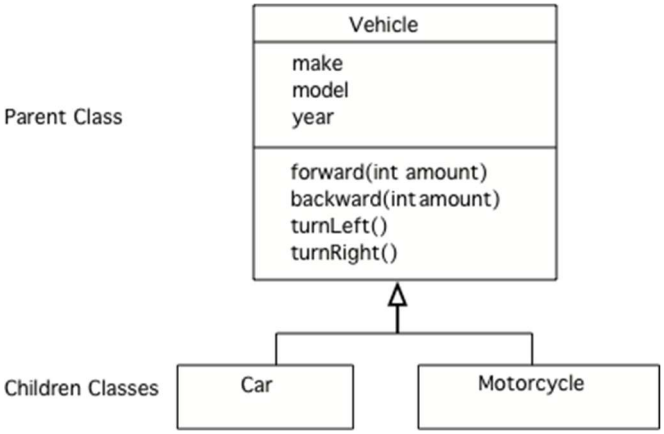

## Example

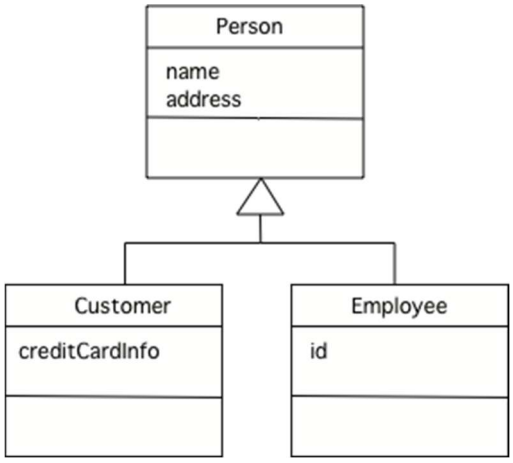


## Or


BankAccount
owner : String
balance : Dollars
deposit ( amount : Dollars )
withdrawal ( amount : Dollars )
CheckingAccount
insufficientFundsFee : Dollars
processCheck ( checkToProcess : Check )
withdrawal ( amount : Dollars )
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal ( amount : Dollars )

## Example

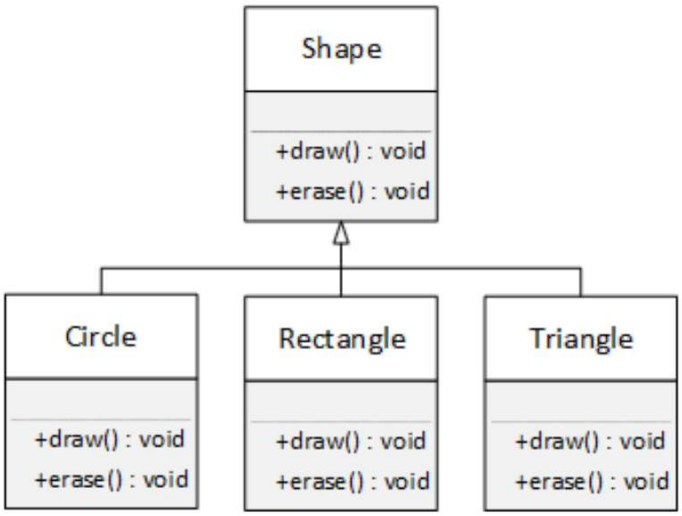

Parent Class
Vehicle
make
model
year
forward(int amount)
backward(int amount)
turnLeft()
turnRight()
Children Classes
Car
Motorcycle


Person
name
address
Customer
creditCardInfo
Employee
id

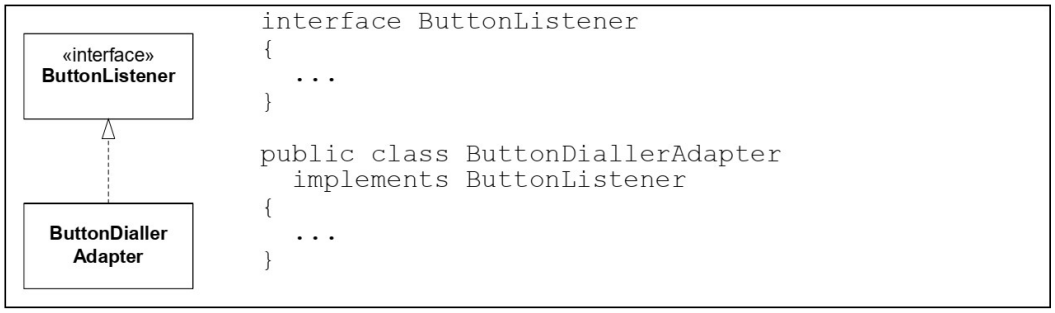
## Polymorphism

### Example

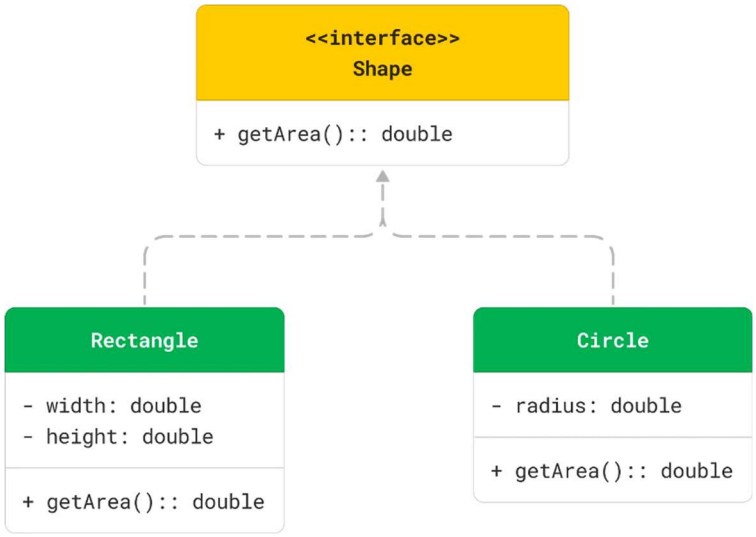
**UML class diagram of the Shape classes**. Polymorphism requires inheritance and an overridden function. Shape is the superclass or parent class; Circle, Rectangle, and Triangle are all subclasses or children of Shape. The drawing functions in the subclasses override the draw function in the **Shape class**.




## interface



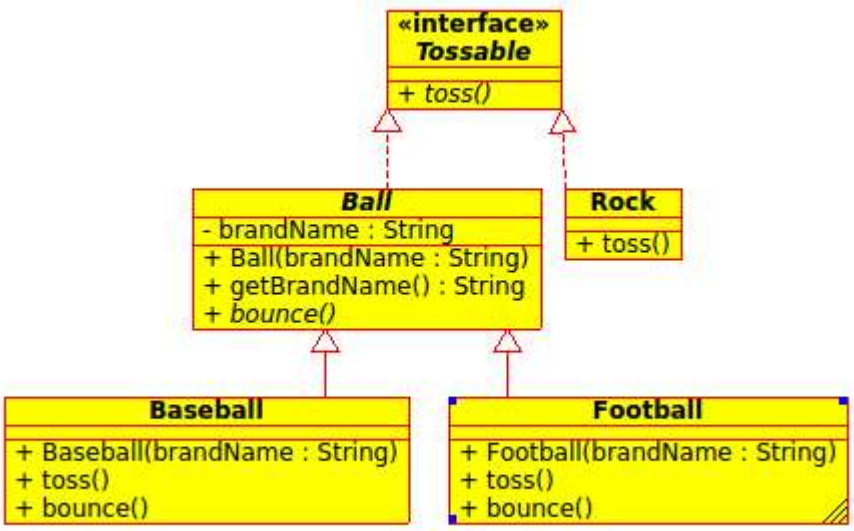

**Example**: A Rectangle class and a Circle class implement the Shape interface, which declares a getArea() method.




## Example

```
public interface Tossable
{  void toss(); }

public abstract class Ball implements Tossable
{ private String brandName;

   public Ball(String brandName)
   {  this.brandName = brandName;    }

   public String getBrandName()
   { return brandName;     }

   public abstract void bounce();  }
   public class Baseball extends Ball {
   public Baseball(String brandName)
   {   super(brandName);      }

   public void toss()
   {
   }

   public void bounce()
   {
   }
```

```
}
public class Football extends Ball
{     public Football(String brandName)
   {   super(brandName);     }

   public void toss()
   {
   }

   public void bounce()
   {
   }
}
public class Rock implements Tossable
{   public void toss()
   {
   } }
```



## Abstract classes

In UML there are **two ways to denote that a class, or a method, is abstract**. You can **write the name in italics**, or you can use the **{abstract}** property.