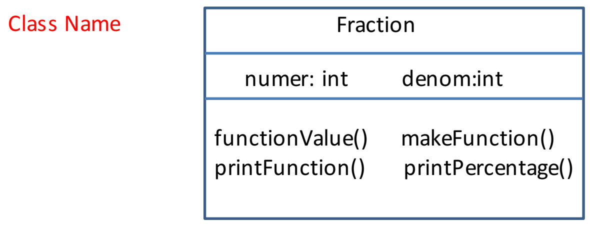
Object-Oriented php: Using Existing classes

Once we have defined a class we may use it in the development of new classes.

Let’s create a new class `TwoFractions` that has two fractions as data and acts on this data is various ways including:

- Comparing two fractions for equality
- Summing two fractions
- Multiplying two fractions
- Subtracting two fractions and
- Dividing two fractions

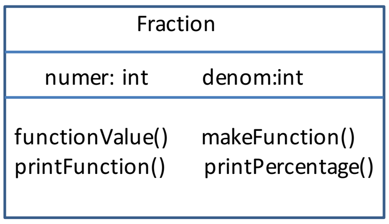
UML: Representing a Class



UML: Representing a Class

Class Name

Data (Class attributes)
These are presented as
name:type

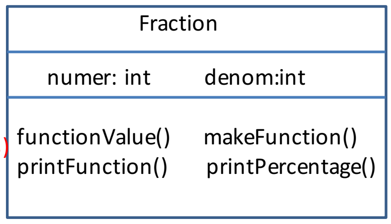


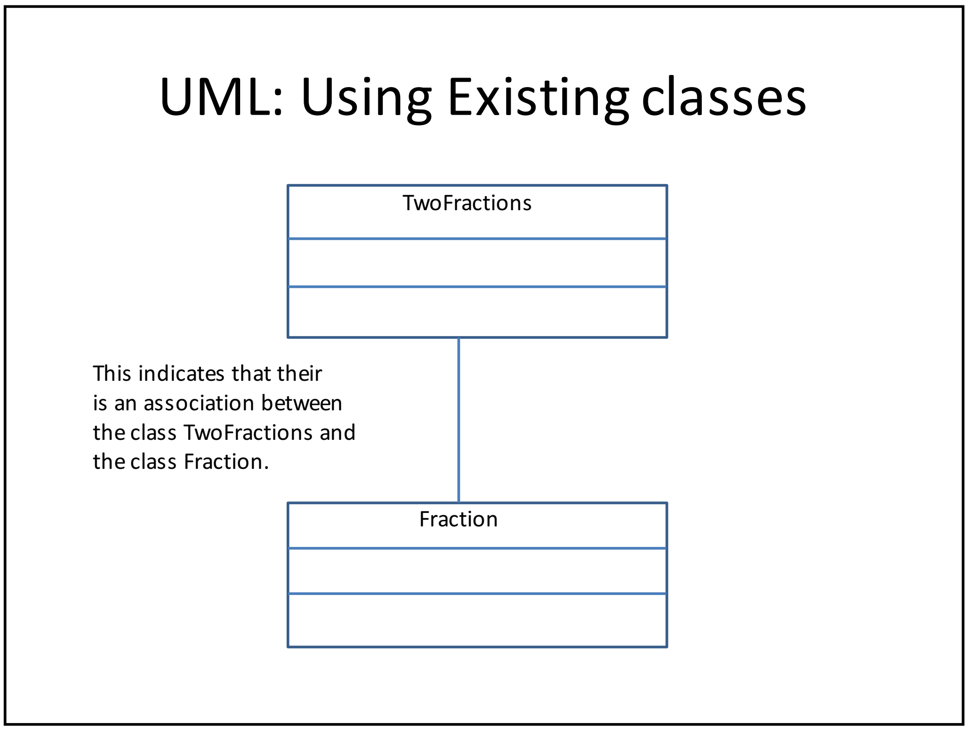
UML: Representing a Class

Class Name

Data (Class attributes)

Functions (Class methods)



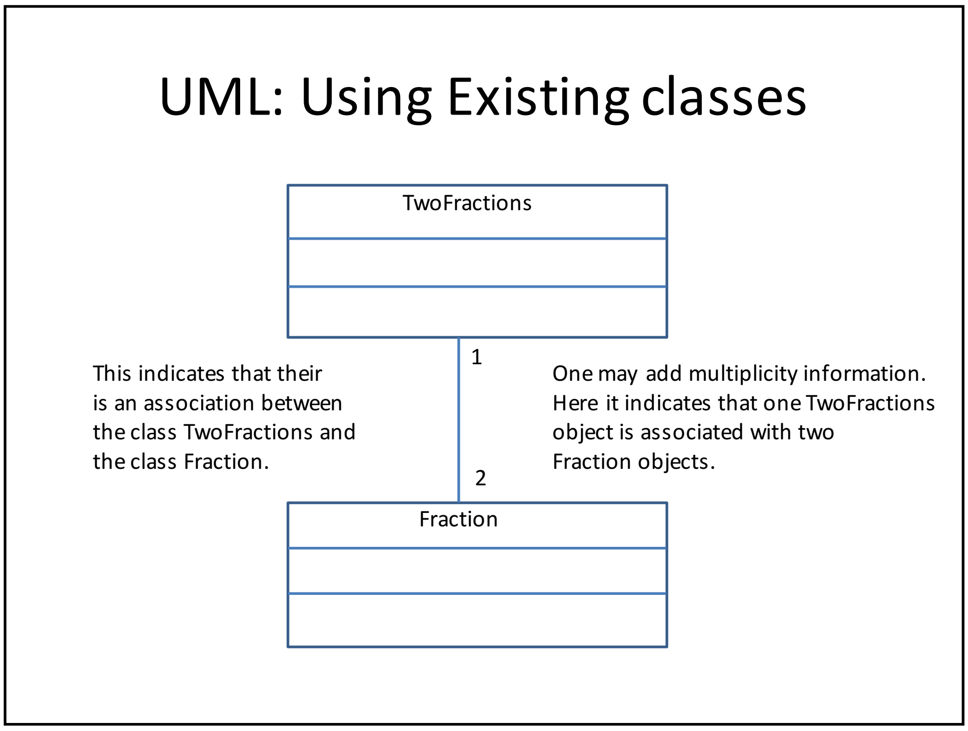


UML: Using Existing classes

TwoFractions

This indicates that their
is an association between
the class TwoFractions and
the class Fraction.

Fraction



UML: Using Existing classes

TwoFractions

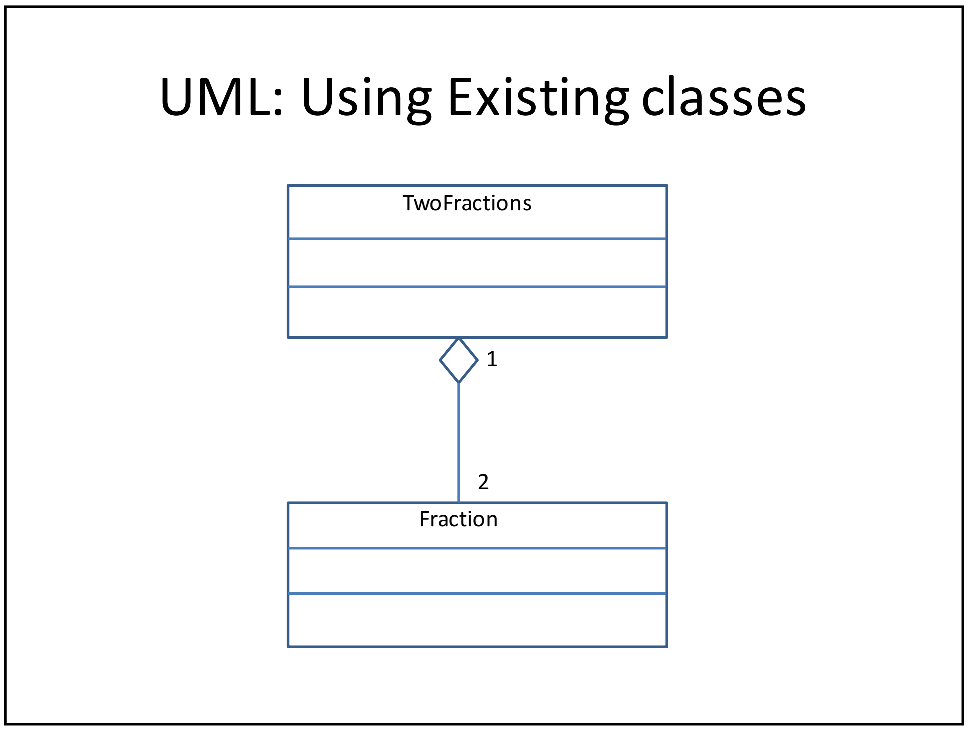
1

This indicates that their
is an association between
the class TwoFractions and
the class Fraction.

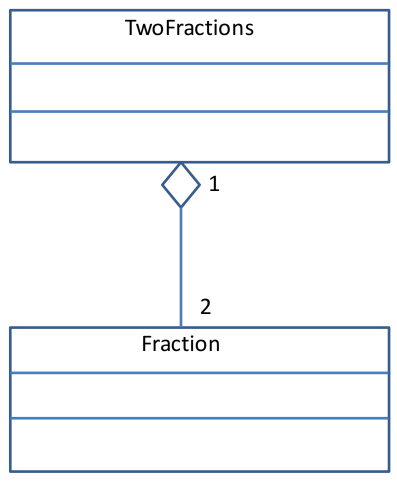
One may add multiplicity information.
Here it indicates that one TwoFractions
object is associated with two
Fraction objects.

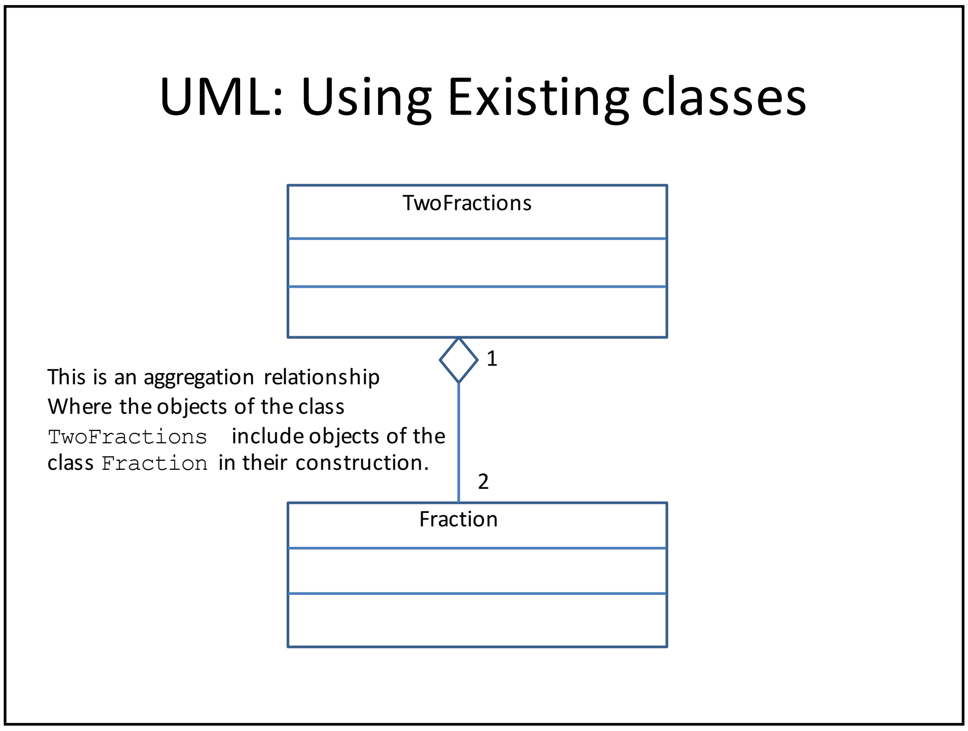
2

Fraction

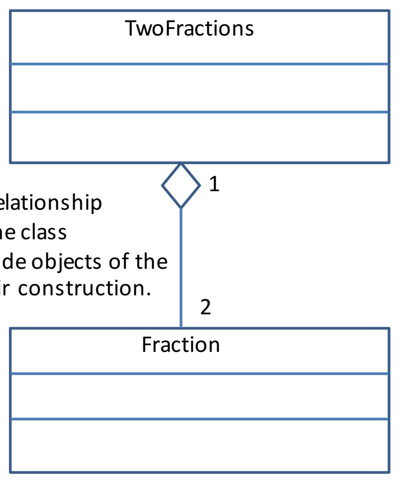


UML: Using Existing classes



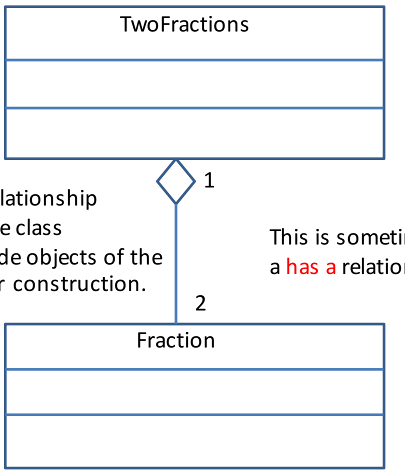


UML: Using Existing classes



This is an aggregation relationship
Where the objects of the class
`TwoFractions` include objects of the
class `Fraction` in their construction.

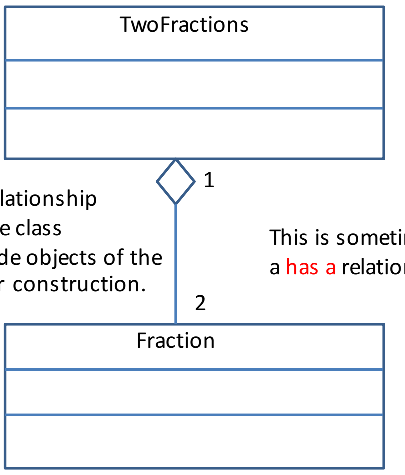
UML: Using Existing classes



This is an aggregation relationship
Where the objects of the class `TwoFractions` include objects of the class `Fraction` in their construction.

This is sometimes referred to as a has a relationship.

UML: Using Existing classes



This is an aggregation relationship
Where the objects of the class `TwoFractions` include objects of the class `Fraction` in their construction.

This is sometimes referred to as a has a relationship.

Note that the Fraction objects exist outside of the TwoFraction objects.

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- One data value – the array of fractions
- A constructor function
- A function to add a Fraction to the array
- A function to print the contents of the array

Exercise

Define the class FractionArray.
Declare four Fraction objects.
Declare a FractionArray object
Add each Fraction object to the FractionArray object.
Print out the contents of the FractionArray object.

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions.

```
include 'fractionOO.php';
```

We want to use the Fraction class which is defined in this file

```
class FractionArray{



}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- One data value – the array of fractions

```
include 'fractionOO.php';

class FractionArray{




}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- One data value – the array of fractions

```
include 'fractionOO.php';

class FractionArray{

       var $array;


}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- A constructor function

```
include 'fractionOO.php';

class FractionArray{

        var $array;


}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- A constructor function

```
include 'fractionOO.php';

class FractionArray{

        var $array;

        function __construct($a){
                $this->array = $a;
        }

}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- A function to add a Fraction to the array

```
include 'fractionOO.php';

class FractionArray{

        var $array;

        function __construct($a){
                $this->array = $a;
        }

}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- A function to add a Fraction to the array

```
include 'fractionOO.php';

class FractionArray{

        var $array;

        function __construct($a){
                $this->array = $a;
        }

        function addElement($a){
                $this->array[] = $a;
        }
}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- A function to print the contents of the array

```
include 'fractionOO.php';

class FractionArray{

        var $array;

        function __construct($a){
                $this->array = $a;
        }

        function addElement($a){
                $this->array[] = $a;
        }
}
```

Object-Oriented php: Classes

Once a class is defined we may use it to create other classes. For example, we could create a class `FractionArray` that represents an array of fractions. It has:

- A function to print the contents of the array

```
include 'fractionOO.php';

class FractionArray{

        var $array;

        function printFractionArray(){
                for($i=0;$i<count($this->array);$i++){
                        $this->array[$i]->printFraction();
                }
        }

}
```

Object-Oriented php: Classes

Object-orientation is a software development approach in which the main building blocks are classes which have both data (aka. variables, attributes or properties) and functions (aka. methods) that act on the data. They are a **blueprint** for one or more objects. That is, each object of the class has the same structure as the class but has values assigned to each of the variables.

For example:

```
class  Fraction {

        var $numer;
        var $denom;
        …
        function fractionValue(){
                return $this->numer / $this->denom;
        }

}
```

Object-Oriented php: Classes

Object-orientation is a software development approach in which the main building blocks are classes which have both data (aka. variables, attributes or properties) and functions (aka. methods) that act on the data. They are a blueprint for one or more objects. That is, each object of the class has the same structure as the class but has values assigned to each of the variables.

For example:

```
class  Fraction {

        var $numer;                         Here is the data…
        var $denom;
        …
        function fractionValue(){
                return $this->numer / $this->denom;
        }

}
```

Object-Oriented php: Classes

Object-orientation is a software development approach in which the main building blocks are classes which have both data (aka. variables, attributes or properties) and functions (aka. methods) that act on the data. They are a blueprint for one or more objects. That is, each object of the class has the same structure as the class but has values assigned to each of the variables.

For example:

```
class  Fraction {

       var $numer;
       var $denom;
       …
       function fractionValue(){
       return $this->numer / $this->denom;

       }
}
```

and here is a function that acts on the data

Object-Oriented php: Classes

Object-orientation is a software development approach in which the main building blocks are classes which have both data (aka. variables, attributes or properties) and functions (aka. methods) that act on the data. They are a blueprint for one or more objects. That is, each object of the class has the same structure as the class but has values assigned to each of the variables.

For example:

```
$frac1 = new Fraction(10,17);
$frac2 = new Fraction(2,5);
```

The __construct function is used to create these objects

`Sfrac1` and `$frac2` are Fraction objects.

`$frac1` has a `$numer` variable with value 10 and a `$denom` variable with value 17.
`$frac2` has a `$numer` variable with value 2 and a `$denom` variable with value 5.

Since they are `Fraction` objects one may call any of the functions defined in the `Fraction` class from these objects. That is, `fractionValue()`, `makeFraction()`, `printFraction()` and `printPercentage()` .

Object-Oriented php:Inheritance

One of the most important features of object-orientation is INHERITANCE.

This is the ability to create new classes from existing classes by EXTENDING THEM.

That is, the new class has the data and functions of the existing class but:

- may have some additional data
- may have some additional functions
- may change the functionality of one or more of the functions of the class that it is extending

Object-Oriented php: Extending Classes

We may create a new class by extending existing classes. We do this if a new class shares the data and behaviour of an existing class. That is, if a new class is more specialised version of an existing class.

For example, there is a subset of fractions referred to as improper fractions. They are fractions but with the numerator greater than the denominator.

That is, an improper fraction **is a** fraction but with an additional characteristic.

Object-Oriented php: Extending Classes

We use the php keyword extends to create a new class with the data and functionality of an existing class.

```
class ImproperFraction extends Fraction {
…
}
```

UML

The Unified Modelling Language (UML) provides graphical notation to represent classes, objects, the associations between classes and the communications between objects.

A class is simply represented as a rectangle with three segments: the top segment presents the name of the class; the middle segment presents the data (properties, attributes) of the class; and the bottom segment presents the functions (methods) of the class.

An association is represented by a line between the associated classes. This may be annotated with information regarding the multiplicity of the association and the nature of the association.

When one class has parts that are an existing class we say that the two classes have a whole/part relationship. There are two types of this:

- aggregation where the ‘part’ objects exist independently of the ‘whole’ objects.
- composition where the existence of the ‘part’ objects depend on the existence of the ‘whole’ object.

UML

The Unified Modelling Language (UML) is a graphical language used to model object-oriented (OO) systems. The benefits of using UML are:

- It provides a pictorial representation of a design. That is, it provides a series of diagrams that highlight the essential features of an OO design.
- It provides a clear medium for communication that doesn't require knowledge of a particular programming language /scripting language. Thus one may present UML diagrams to a client who would be able to understand the design without the need to look at programming / scripting code. It therefore focuses on the essential features of the design (removing all of the syntactic detail of the code).
- It provides a permanent record of a design. That is, it exists independently of the implemented system and can be used in future to build similar systems.
- UML diagrams are independent of any particular programming/scripting language and therefore may be implemented in any OO language.

UML

When one class has parts that are an existing class we say that the two classes have a whole/part relationship. There are two types of this:

- aggregation where the 'part' objects exist independently of the 'whole' objects

  An example of this is a FractionArray has a collection of Fractions. These Fractions exist outside of the FractionArray but are used to create a FractionArray.

- composition where the existence of the 'part' objects depend on the existence of the 'whole' object.

  An example of this is a University that has a collection of faculties. The faculties only exist as part of the University.