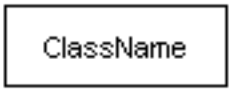
Class Diagrams

Visualising a Class

In programming things fall into categories called **classes**. The UML class diagram consists of several classes connected with relationships. But how does a UML represent a class?

The rectangle is the icon for the class. The name of the class is, by convention, a word with an initial uppercase letter. It appears near the top of the rectangle. If your class name has more than one word name, then join the words together and capitalize the first letter of the every word.



The **Class name** is written at the top of the rectangle

Class Atrributes

An **attribute** is a property of a class. It describes a range of values that the property may hold in objects of the class. A class may have zero or more attributes.

A one-word attribute name is written in lowercase letter. If the name consists of more than one word, the words are joined and each word other than the first word begins with an uppercase letter. The list of attribute names begins below a line separating them from the class name.

ClassName
attribute1 : type1 attribute2 : type2 = “Default value”

MyClass
firstName : String teacherName : String = “Ms Dyke”

In the class icon, you can specify a type for each attribute’s value (String, double, int etc.). Also, you can indicate a default value for an attribute.

An **operation** is something that a class can do, or that you (or another class) can do to a class.

Class Behaviours

Like an attribute name, an operation's name is written in lowercase letters. If the name consists of more than one word, the words are joined and each word except the first word begins with an uppercase letter. The list of operations begins below a line separating operations from the attributes.

Basic Template

ClassName
attribute1 : type1 attribute2 : type2 = “Default value”
operation1 ( ) operation2 (parameter list) operation3 ( ) : return value type

Example Class

MyClass
firstName : String teacherName : String = “Ms Dyke”
getFirstName ( ) setGradeLevel (int grade) calculateAge ( ) : int

Operations may have some additional information in the parentheses that follow an operation name. You can show the parameter pass followed by the return type.

Other additional features that can be attached to attributes of the class are **constraints** and notes. Constraints are free-form text enclosed in braces. The bracketed text specifies one or more rules the class follows.

Notes usually are attached to attributes as well as operations, and they give additional information to a class. A note can contain a graphic as well as text.

How can we derive classes from interviewing the clients?

In conversations with clients, be alert to the nouns they use to describe the entities in their business. Those nouns will become the classes in your model. Be alert also to the verbs that you hear, because these will constitute the operations in those classes. The attributes will emerge as nouns related to the class nouns.

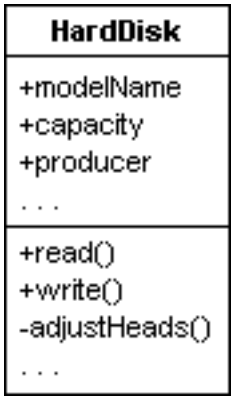
Visibility

Visibility applies to attributes or operations, and specifies the extent to which other classes can use a given class's attributes or operations.

Three levels of visibility are possible (the symbols are used in UML classes to indicate different levels of visibility):

Symbol	Level
+	public level - usability extends to other classes
#	protected level - usability is open only to classes that inherit from original class
-	private level - only the original class can use the attribute or operation

Public and Private operations in a Hard Disk



Class Diagram - Example

Here is a brief description for a writing a text document:

Suppose that you're writing a document using some famous text processing tools, like Microsoft Word for example. You can start typing a new document, or open an existing one. You type text by using your keyboard.

Every document consists of several pages, and every page consists of a header, document body and/or a footer. In the header and footer you may add a date, time, page number, file location etc..

The document's body has sentences. Sentences are made up of words and punctual signs. Words consist of letters, numbers and/or special characters. You may also insert pictures and tables. Tables consist of rows and columns. Every cell from a table may hold either text or pictures.

After finishing the document, user can choose to save or to print the document.

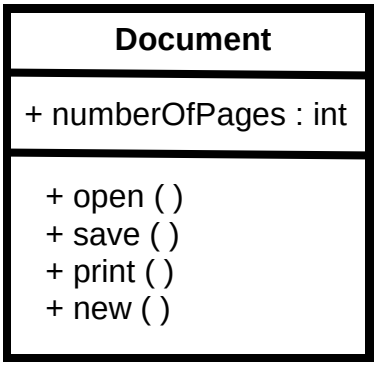
This is simplified explanation of creating a text document. If we extract the list of nouns from the previous text, the following list can be achieved:

document, text processing tool, Microsoft Word, text, keyboard, header, footer, document body, date, time, page number, location of file, page, sentence, word, punctual sign, letter, number, special character, picture, table, row, column, cell, user

Nouns coloured in red are candidate classes, and candidate attributes for our model.

Let's start with the document. As you can see, this example deals with documents, thus **Document** will be the central class in our class diagram. Document has a several pages, therefore numberOfPages will be one of the attributes for the Document class. For the operations we have: open( ), save( ), print( ) and new( ). Every document consists of pages. **Page** will also become a class.

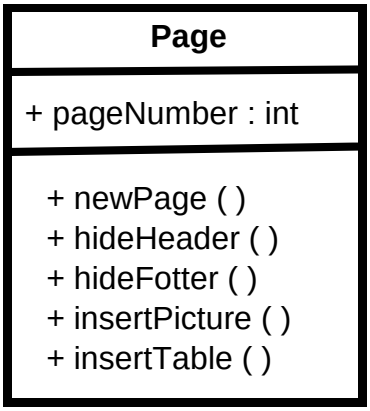
The Document class:



The Page class will hold pageNumber as an attribute, and operations allowed here can be:newPage( ), hideHeader( ) and hideFooter( )

Operations for the header and footer tells us that **Header** and **Footer** can also be classes.

The Page class:

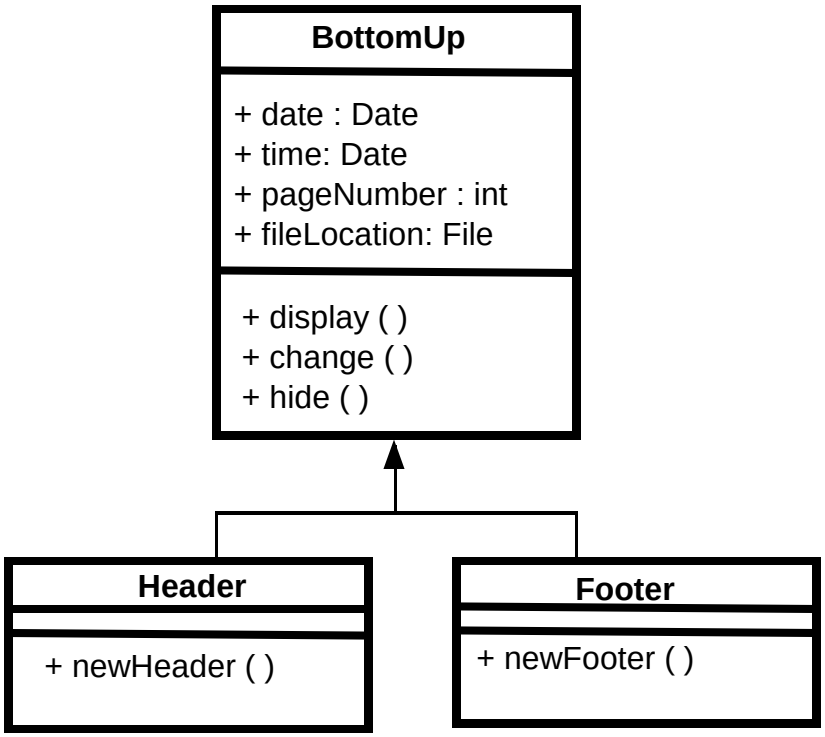


The Header class and the Footer class have common attributes: date, time, pageNumber and fileLocation. These attributes are optional for every header or footer and the user may configure them. This will demonstrate that a common class can be introduced and provide an opportunity to demonstrate inheritance.

The parent class will be **BottomUp** (this name is chosen because headers and footer appear in upper and bottom parts of every page) and will hold common attributes for Header and Footer, and these operations: display( ), hide( ) and change( ).

Header and Footer classes (children of this class) will have only these operations: newHeader( ) and newFooter( )

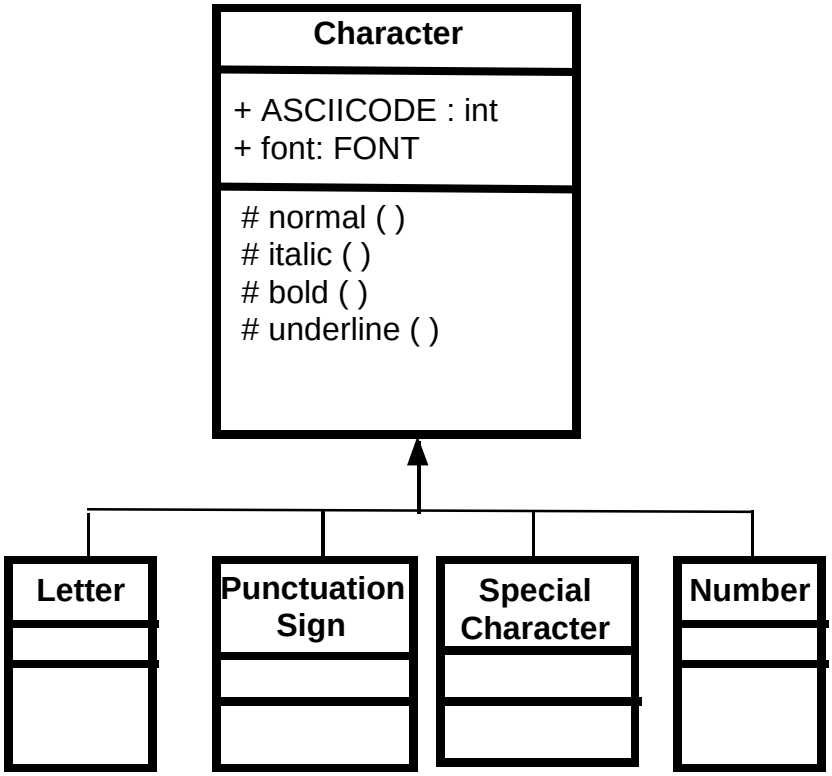
The BottomUp class and it's children:



:

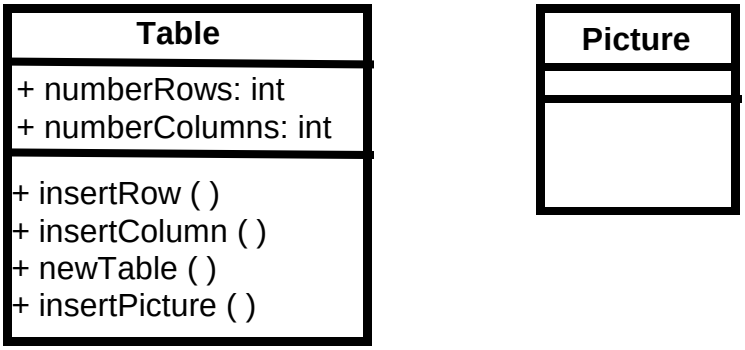
Before examining the document's body or text, let’s take a look at the making of text actions. A Document's text is made up of sentences. Sentences are made up of words and words are made up of characters. If words are array of characters and sentence is array of words, then a sentence is also an array of characters. Therefore a document's body can be an array of characters. For this purpose to make a Document's text we'll use the Character class with its children. The Character class will have ASCII code and type as attributes (type tells the type of the character - normal, italic, bold or underline), and normal( ), bold( ), italic( ) and underline( ) as operations. The Character class children will be: Letter, PunctualSign, SpecialCharacter and Number. Remember that tables and pictures can be found in the document's body. Here are new classes in our class diagram.

The Character class hierarchy:

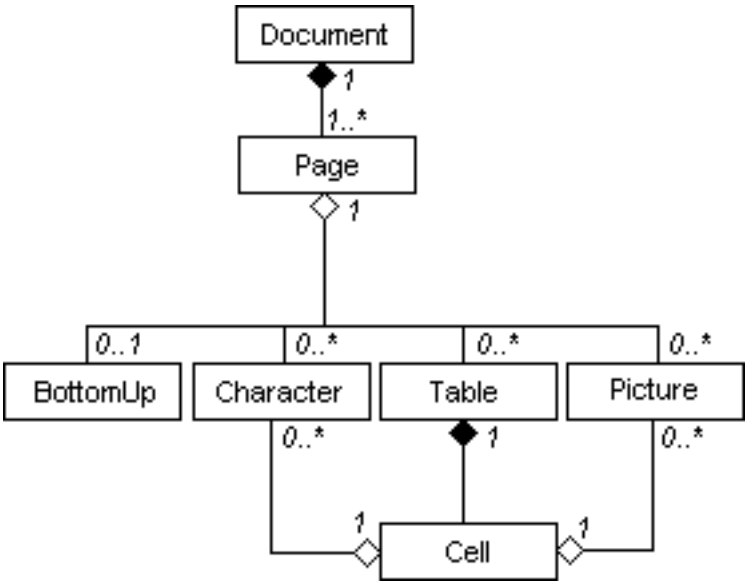


The **Table** class has numbRows and numbColumns as attributes and newRow( ), newColumn( ) and newTable( ) as operations. Every table consists of one or more cells. And in every cell, text or pictures can be placed.

The **Table** class and the **Picture** class



Having all these classes in front of us, we can draw out the associations between them and gain the completed class diagram for this problem.



The class diagram for the making of text document. Please note that for the purposes of this diagram the Class UMLs have been limited to the class name only!!! Normally, each Class UML would be shown - name, attributes and behaviours - with their access levels all on one page. The lines would then be added to demonstrate the links between the classes. Notice the small annotations above/below each class name. *1..** means that at least 1 object would be created of that particular class. *0..** means zero to n instances would be created, and so on.

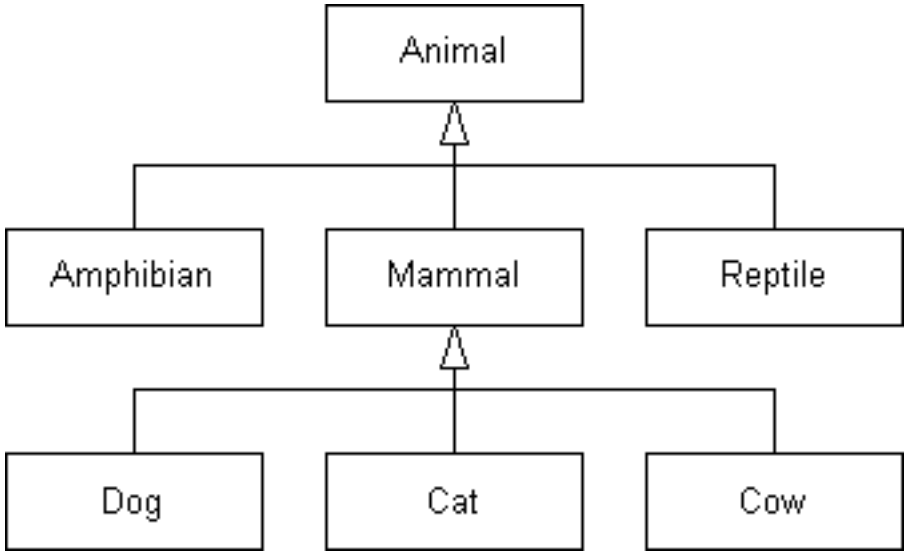
The class diagram above is not finished in detail. If you know more about typing a document, then add the information you know to the given diagram. This model has the potential to grow and grow, and making all the necessary steps of analysis and design you may reach a point where a new text processing tool is modelled.

Hierarchichal Tree = Inheritance & Generalization

If you know something about a category of things, you automatically know some things you can transfer to other categories.

If you know something is an animal, you take for granted that it eats, sleeps, has a way of being born, has a way of getting from one place to another... But imagine that mammals, amphibians and reptiles are all animals. Also cows, dogs, cats... are grouped in category mammals. Object-orientation refers to this as inheritance.

Animal Ineritance Hierarchy



One class (the child class or subclass) can **inherit** attributes and operations from another (the parent class or superclass). The parent class is more general than the child class.

In general, a child is substitutable for a parent. That is, anywhere the parent appears, the child may appear. The reverse isn't true, however.

In a UML, inheritance is represented with a line that connects the parent to a child class, and on the parent's side you put an open triangle.

If we look from the association's side, the inheritance is a kind of association.

What should analysts do to discover inheritance?

The analyst has to realize that the attributes and operations of one class are general and apply to perhaps several other classes - which may add attributes and operations of their own. Another possibility is that the analyst notes that two or more classes have a number of common attributes and operations.

Classes that provide no objects are said to be **abstract classes**. You indicate an abstract class by writing its name in italics.

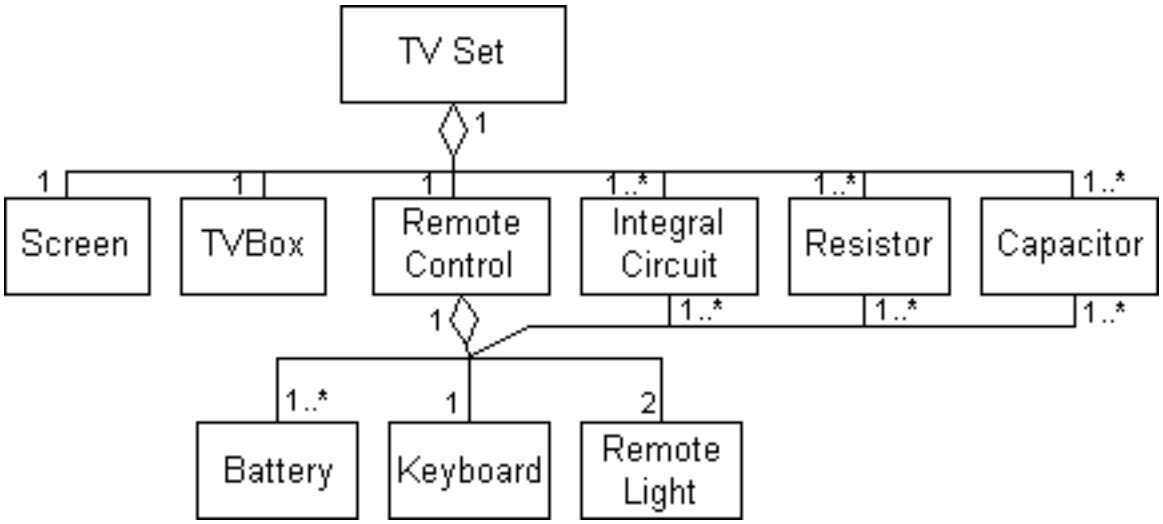
A class can inherit attributes and operations from another class. The inheriting class is the child of the parent class it inherits from. Abstract classes are intended only as bases for inheritance and provide no objects of their own.

Aggregations

Sometimes a class consists of a number of component classes. This is a special type of relationship called an aggregation. The components and the class they constitute are in a part-whole association.

Aggregation is represented as a hierarchy with the "whole" class at the top, and the component below. A line joins a whole to a component with an open diamond on the line near the whole.

Let's take a look at the parts in a TV set. Every TV has a TV box, screen, speaker(s), resistors, capacitors, transistors, ICs... and possibly a remote control. A remote control can have these parts: resistors, capacitors, transistors, ICs, battery, keyboard and remote lights.

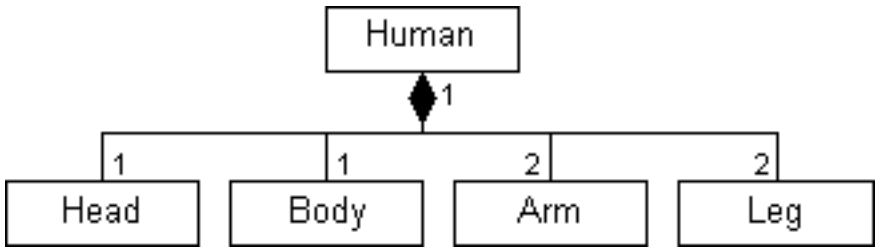


An aggregation association in the TVSet system.

Sometimes the set of possible components in an aggregation falls into an OR relationship. To model this, you would use a constraint - the word OR within braces on a dotted line that connects the two part-whole lines.

A **composite** is a strong type of aggregation. Each component in a composite can belong to just one whole. The symbol for a composite is the same as the symbol for an aggregation except the diamond is filled.

If you examine the human's outside you'll find out that every person has the following: head, body, arms and legs. This is shown on this picture.



A composite association where each component belongs to exactly one whole.

An aggregation specifies a part-whole association. A "whole" class is made up of component classes. A composite is a strong form of aggregation, and a component in a composite can be part of only one whole. Aggregations and composites are represented as lines joining the whole and the component with open and filled diamond, respectively, on the whole side.

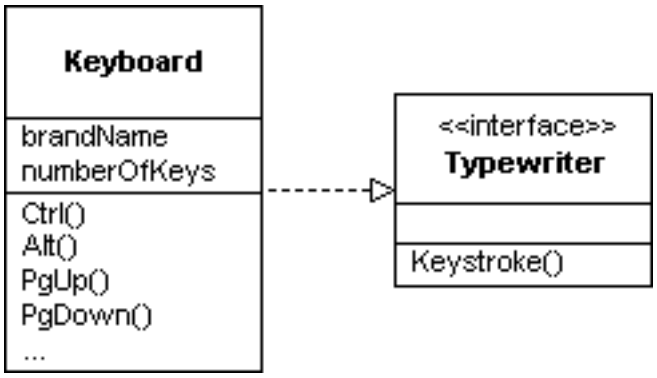
Interfaces and Realizations

In previous lessons we learned how to refine classes from the interview with client, and define the different relationships between them. But it's possible that some of classes are not related to a particular parent, but their behaviours might include some of the same operations with the same signatures. You can code the operations for one of the classes and reuse them in the others.

An **interface** is a set of operations that specifies some aspect of a class behaviour, and it's a set of operations a class presents to other classes.

You model an interface the same way you model a class, with the rectangle icon, but an interface has no attributes, only operations. Another way is with a small circle joined with a line to a class.

The computer's keyboard is a reusable interface. Its keystroke operation has been reused from the typewriter. The placement of keys is the same as on a typewriter, but the main point is that the keystroke operation has been transferred from one system to another. Also on a computer's keyboard you'll find a number of operations that you won't find on a typewriter (Ctrl, Alt, PageUp, PageDown...etc.)



An interface is a collection of operations that a class carries out.

To distinguish interfaces from classes, in stereotype construct we put <<interface>> or **I** at the beginning of the name of any interface.

The relationship between a class and an interface is called a **realization**. This relationship is modelled as a dashed line with a large open triangle adjoining and pointing to the interface.