IBM

UML basics: The class diagram

An introduction to structure diagrams in UML 2

from The Rational Edge: As the most important example of the new structure diagram diagram can be used by analysts, business modelers, developers, and testers through lifecycle. This article offers a comprehensive introduction.

Donald Bell is an IT architect for IBM Rational software, where he works with clients to define and adop and tools.

UML basics: The class diagram

An introduction to structure diagrams in UML 2

nt in a series of articles
ams used within the
ge, or UML. In my
previous article on sequence diagrams, I shifted
focus away from the UML 1.4 spec to OMG's
on of UML (a.k.a.
iscuss Structure Diagrams, which is a new
een introduced in UML 2. Because the purpos
ements and their meanings, this article focuse
become clear. Subsequent articles will cover
structure category.

This is the next installment in a series of articles about the essential diagrams used within the Unified Modeling Language, or UML. In my previous article on sequence diagrams[1], I shifted focus away from the UML 1.4 spec to OMG's Adopted 2.0 Draft

I also want to remind readers that this series is about UML notation eleme Modeling Language, or UML. There are many approaches to modeling, or how should be modeled in the first place. Instead, the purpose of this article ar

help with a basic understanding of notation elements — their syntax and t knowledge you should be able to read diagrams and create your own diag elements.

This article assumes you have a rudimentary understanding of object-orie who need a little assistance with OO concepts, you might try the Sun brie Programming Concepts. *Reading the sections* "What Is a Class?" *and* "W *you enough understanding to make this article useful. In addition, David T* Technologies: A Manager's Guide*, offers an excellent, high-level explanat without requiring an in-depth understanding of computer programming.*

The yin and yang of UML 2

In UML 2 there are two basic categories of diagrams: structure diagrams and behavior diagrams. Every UML diagram belongs to one these two diagram categories. The purpose of structure diagrams is to show the static structure of the system being modeled. They include the class, component, and or object diagrams. Behavioral diagrams, on the other hand, show the dynamic behavior between the objects in the system, including things like their methods, collaborations, and activities. Example behavior diagrams are activity, use case, and sequence diagrams.

Re
F
U
U
m
W
IE
D
th
V
IE
C
B

Structure diagrams in general

As I have said, structure diagrams show the static structure of the system elements of a system, irrespective of time. Static structure is conveyed by instances in the system. Besides showing system types and their instance show at least some of the relationships among and between these eleme their internal structure.

Structure diagrams are useful throughout the software lifecycle for a varie
these diagrams allow for design validation and design communication bet
For example, business analysts can use class or object diagrams to mod
and resources, such as account ledgers, products, or geographic hierarch
component and deployment diagrams to test/verify that their design is sou
diagrams to design and document the system's coded (or soon-to-be-cod

The class diagram in particular

UML 2 considers structure diagrams as a classification; there is no diagra
Diagram." However, the class diagram offers a prime example of the struc
provides us with an initial set of notation elements that all other structure
class diagram is so foundational, the remainder of this article will focus or
set. By the end of this article you should have an understanding of how to
and have a solid footing for understanding other structure diagrams when

The basics

As mentioned earlier, the purpose of the class diagram is to show the type
system. In most UML models these types include:

- a class
- an interface
- a data type
- a component.

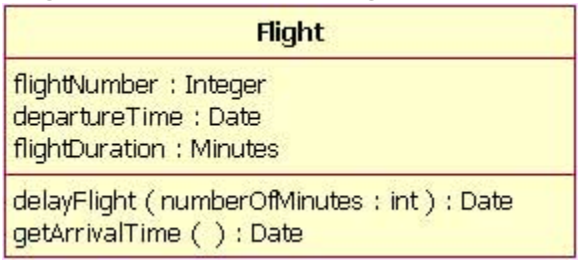
UML uses a special name for these types: "classifiers." Generally, you ca
but technically a classifier is a more general term that refers to the other t

Class name

The UML representation of a class is a rectangle containing three compar

shown in Figure 1. The top compartment shows the class's name. The mi
class's attributes. The bottom compartment lists the class's operations. W
a class diagram, you must use the top compartment, and the bottom two
(The bottom two would be unnecessary on a diagram depicting a higher l
purpose is to show only the relationship between the classifiers.) Figure 1
as a UML class. As we can see, the name is *Flight*, and in the middle com
class has three attributes: flightNumber, departureTime, and flightDuratio
we see that the Flight class has two operations: delayFlight and getArriva

Figure 1: Class diagram for the class Flight



Class attribute list

The attribute section of a class (the middle compartment) lists each of the
line. The attribute section is optional, but when used it contains each attri
list format. The line uses the following format:

```
name : attribute type
```

```
flightNumber : Integer
```

Continuing with our Flight class example, we can describe the class's attri
information, as shown in Table 1.

Table 1: The Flight class's attribute names with their associated type

Attribute Name	Attribute Type
flightNumber	Integer
departureTime	Date
flightDuration	Minutes

In business class diagrams, the attribute types usually correspond to units readers of the diagram (i.e., minutes, dollars, etc.). However, a class diag generate code needs classes whose attribute types are limited to the type language, or types included in the model that will also be implemented in

Sometimes it is useful to show on a class diagram that a particular attribu example, in a banking account application a new bank account would sta UML specification allows for the identification of default values in the attrib following notation:

```
name : attribute type = default value
```

For example:

```
balance : Dollars = 0
```

Showing a default value for attributes is optional; Figure 2 shows a Bank A called *balance*, which has a default value of 0.

Figure 2: A Bank Account class diagram showing the balance attribu dollars

Class operations list

The class's operations are documented in the third (lowest) compartment rectangle, which again is optional. Like the attributes, the operations of a format, with each operation on its own line. Operations are documented u

```
name(parameter list) : type of value returned
```

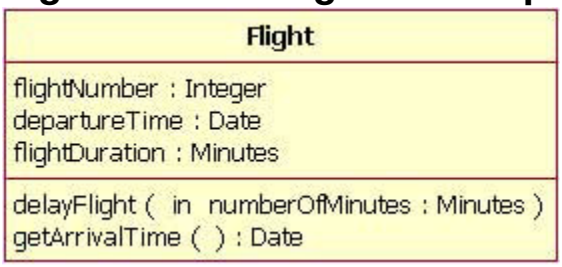
The Flight class's operations are mapped in Table 2 below.

Table 2: Flight class's operations mapped from Figure 3

Operation Name	Parameters Return
delayFlight	**Name** numberOfMinutes
getArrivalTime	N/A

Figure 3 shows that the delayFlight operation has one input parameter — type Minutes. However, the delayFlight operation does not have a return v does not have a return value because I made a design decision not to hav the delay operation should return the new arrival time, and if this were the would appear as `delayFlight(numberOfMinutes : Minutes) : Da` parameters, they are put inside the operation's parentheses; each parame name : parameter type".

Figure 3: The Flight class operations parameters include the optiona



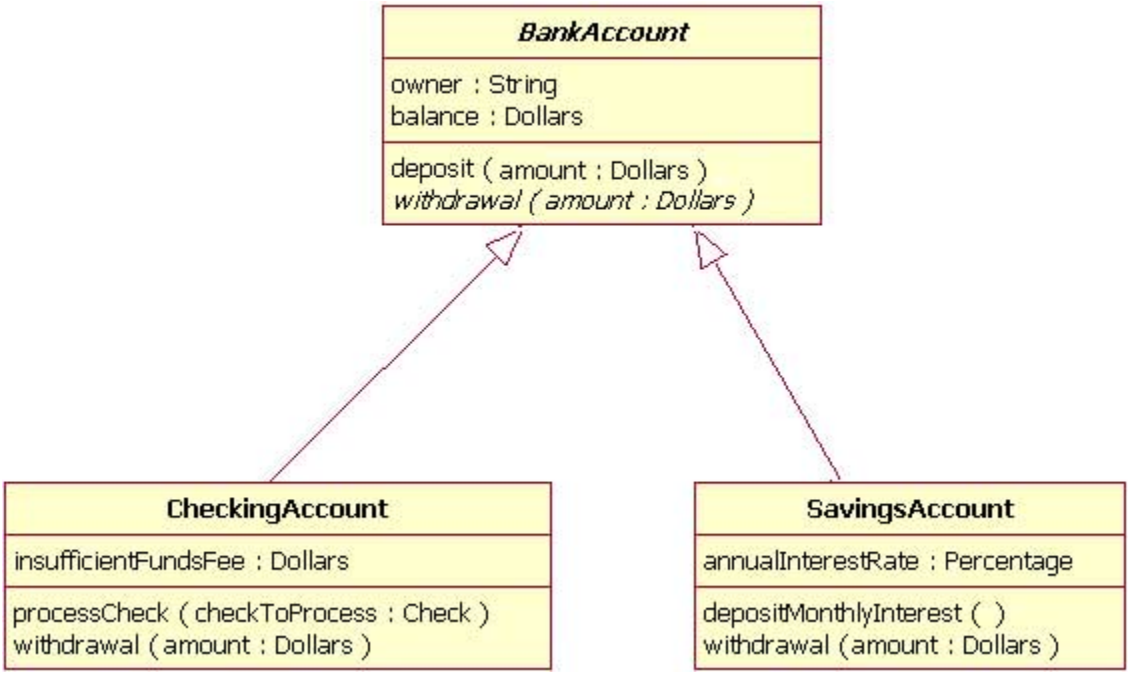
When documenting an operation's parameters, you may use an optional i the parameter is input to, or output from, the operation. This optional indic as shown in the operations compartment in Figure 3. Typically, these indic an older programming language such as Fortran will be used, in which ca helpful. However, in C++ and Java, all parameters are "in" parameters an default type according to the UML specification, most people will leave ou

Inheritance

A very important concept in object-oriented design, *inheritance*, refers to t

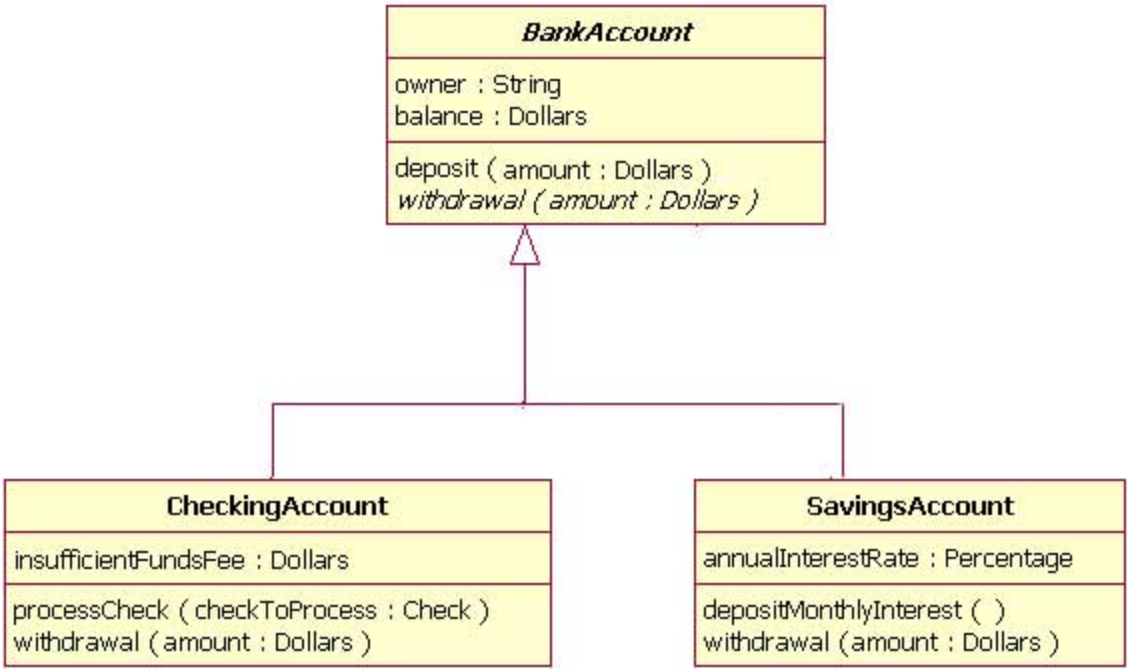
class) to *inherit* the identical functionality of another class (super class), a its own. (In a very non-technical sense, imagine that I inherited my mothe in my family I'm the only one who plays electric guitar.) To model inheritan line is drawn from the child class (the class inheriting the behavior) with a triangle) pointing to the super class. Consider types of bank accounts: Fig CheckingAccount and SavingsAccount classes inherit from the BankAcco

Figure 4: Inheritance is indicated by a solid line with a closed, unfille super class



In Figure 4, the inheritance relationship is drawn with separate lines for ea method used in IBM Rational Rose and IBM Rational XDE. However, ther inheritance called *tree notation*. You can use tree notation when there are Figure 4, except that the inheritance lines merge together like a tree bran the same inheritance shown in Figure 4, but this time using tree notation.

Figure 5: An example of inheritance using tree notation



Abstract classes and operations

The observant reader will notice that the diagrams in Figures 4 and 5 use BankAccount class name and withdrawal operation. This indicates that th abstract class and the withdrawal method is an abstract operation. In othe provides the abstract operation signature of withdrawal and the two child and SavingsAccount each implement their own version of that operation.

However, super classes (parent classes) do not have to be abstract class class to be a super class.

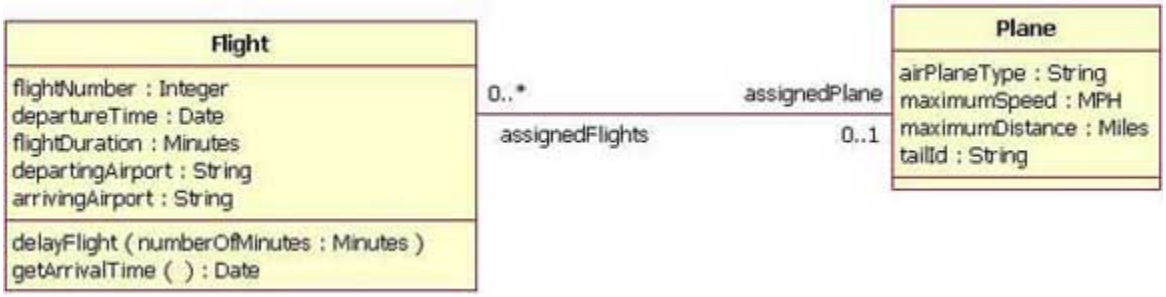
Associations

When you model a system, certain objects will be related to each other, a themselves need to be modeled for clarity. There are five types of associa — bi-directional and uni-directional associations — in this section, and I w association types in the *Beyond the basics* section. Please note that a de each type of association is beyond the scope of this article. Instead, I will association type and show how the association is drawn on a class diagra

Bi-directional (standard) association

An association is a linkage between two classes. Associations are always this means that both classes are aware of each other and their relationshi association as some other type. Going back to our Flight example, Figure association between the Flight class and the Plane class.

Figure 6: An example of a bi-directional association between a Flight



A bi-directional association is indicated by a solid line between the two cla you place a role name and a multiplicity value. Figure 6 shows that the Fli Plane, and the Flight class knows about this association. The Plane takes in this association because the role name next to the Plane class says so the Plane class of 0..1 means that when an instance of a Flight exists, it c a Plane associated with it or no Planes associated with it (i.e., maybe a pl assigned). Figure 6 also shows that a Plane knows about its association v association, the Flight takes on the role of "assignedFlights"; the diagram Plane instance can be associated either with no flights (e.g., it's a brand n infinite number of flights (e.g., the plane has been in commission for the la

For those wondering what the potential multiplicity values are for the ends lists some example multiplicity values along with their meanings.

Table 3: Multiplicity values and their indicators

Potential Multiplicity Values

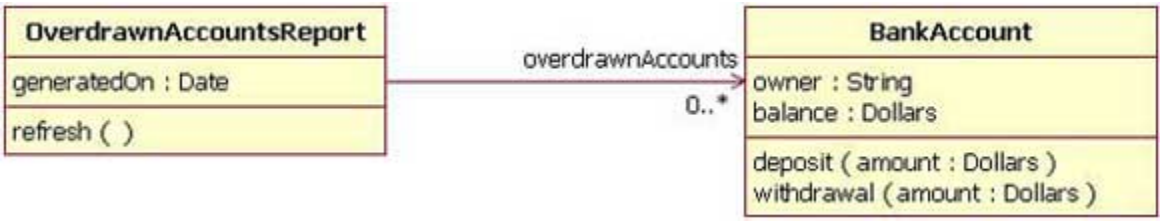
Indicator	Meaning
0..1	Zero or one
1	One only

Indicator	Meaning
0..*	Zero or more
*	Zero or more
1..*	One or more
3	Three only
0..5	Zero to Five
5..15	Five to Fifteen

Uni-directional association

In a uni-directional association, two classes are related, but only one clas exists. Figure 7 shows an example of an overdrawn accounts report with

Figure 7: An example of a uni-directional association: The Overdraw knows about the BankAccount class, but the BankAccount class do association



A uni-directional association is drawn as a solid line with an open arrowhe or triangle, used to indicate inheritance) pointing to the known class. Like uni-directional association includes a role name and a multiplicity value, b bi-directional association, the uni-directional association only contains the value for the known class. In our example in Figure 7, the OverdrawnAcc BankAccount class, and the BankAccount class plays the role of "overdra standard association, the BankAccount class has no idea that it is associa OverdrawnAccountsReport. [Note: It may seem strange that the BankAcc the OverdrawnAccountsReport class. This modeling allows report classes class they report, but the business classes do not know they are being re

coupling of the objects and therefore makes the system more adaptive to

Packages

Inevitably, if you are modeling a large system or a large area of a busines classifiers in your model. Managing all the classes can be a daunting task organizing element called a *package*. Packages enable modelers to orga namespaces, which is sort of like folders in a filing system. Dividing a syst makes the system easier to understand, especially if each package repres system. [Note: Packages are great for organizing your model's classes, b that your class diagrams are supposed to easily communicate informatior modeled. In cases where your packages have lots of classes, it is better t class diagrams instead of just producing one large class diagram.]

There are two ways of drawing packages on diagrams. There is no rule fc use, except to use your personal judgement regarding which is easiest to are drawing. Both ways begin with a large rectangle with a smaller rectan corner, as seen in Figure 8. But the modeler must decide how the packag as follows:

If the modeler decides to show the package's members within the large r members need to be placed within the rectangle. [Note: It's important to those members," I mean only the classes that the current diagram is goi a package with contents does not need to show all its contents; it can sh elements according to some criterion, which is not necessarily all the pa package's name needs to be placed in the package's smaller rectangle (

If the modeler decides to show the package's members outside the large that will be shown on the diagram need to be placed outside the rectang belong to the package, a line is drawn from each classifier to a circle tha circle attached to the package (Figure 9).

Figure 8: An example package element that shows its members insic

boundaries

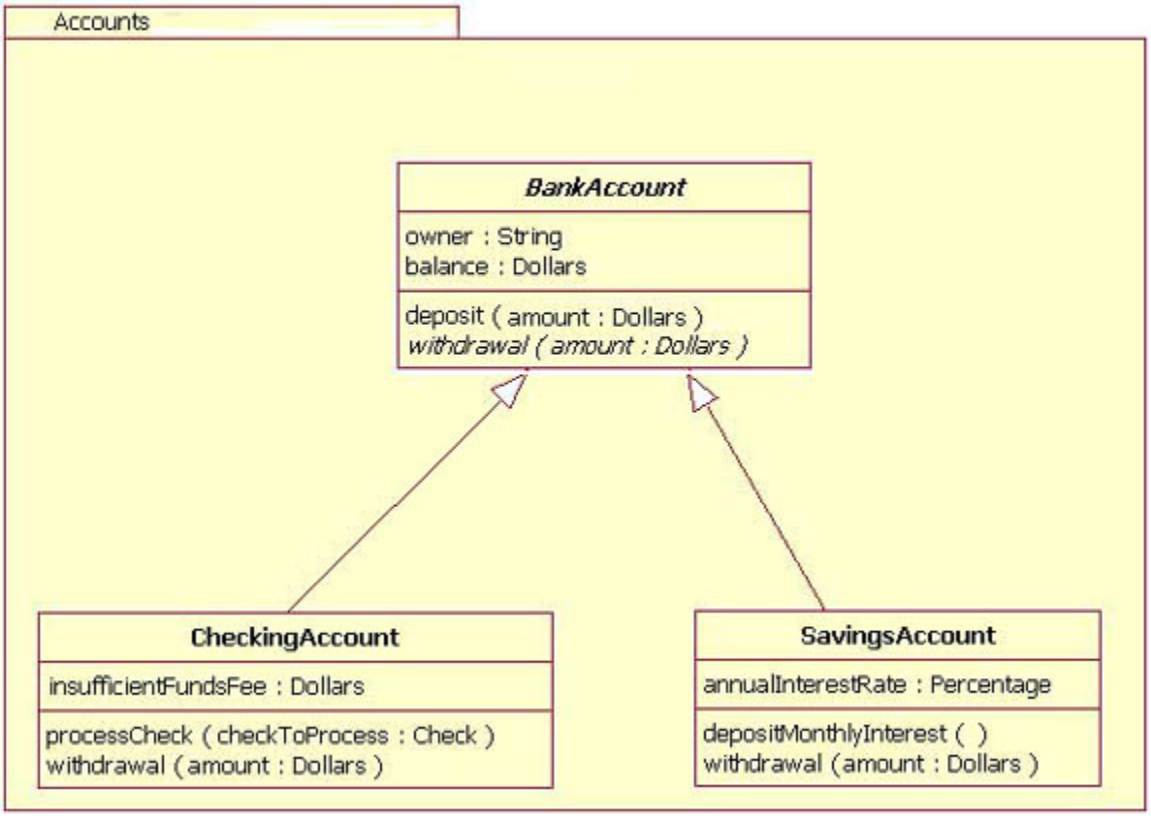
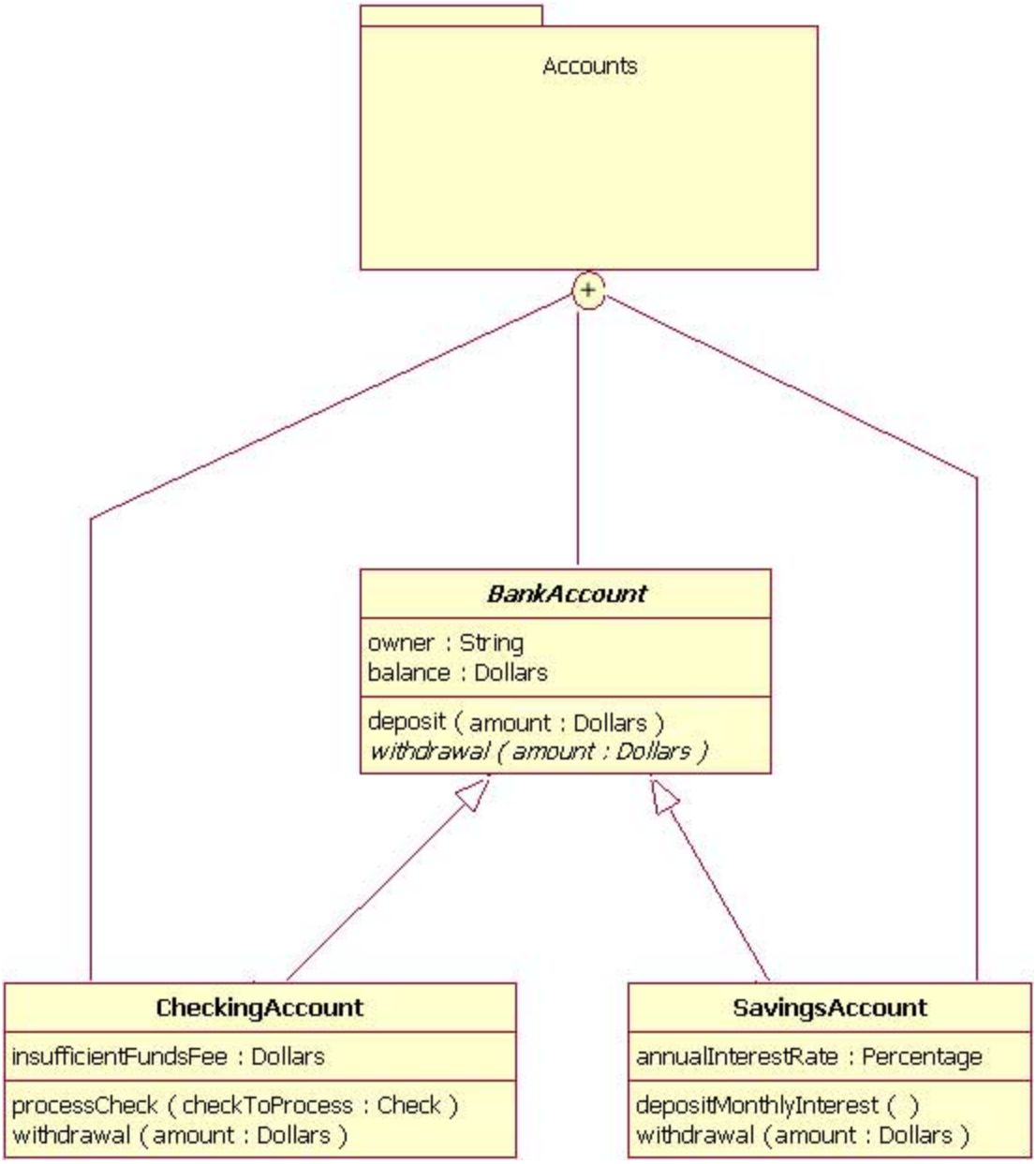


Figure 9: An example package element showing its membership via



Importance of understanding the basics

It is more important than ever in UML 2 to understand the basics of the cl the class diagram provides the basic building blocks for all other structure component or object diagrams (just to name a few).

Beyond the basics

At this point, I have covered the basics of the class diagram, but do not st

sections, I will address more important aspects of the class diagram that y include interfaces, the three remaining types of associations, visibility, and specification.

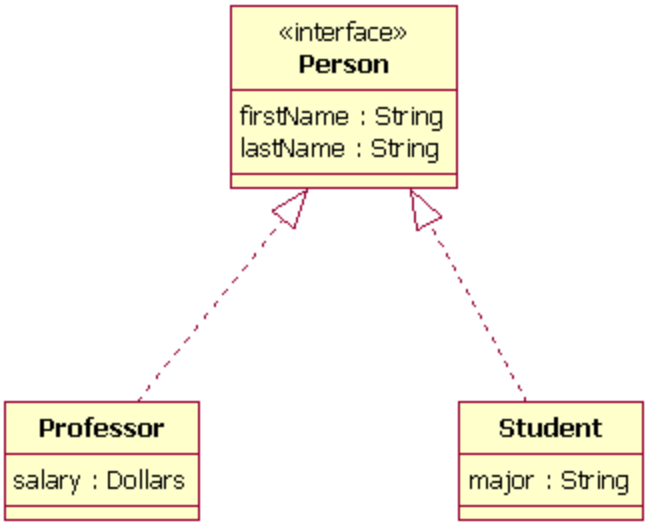
Interfaces

Earlier in this article, I suggested that you think of *classifiers* simply as cla more general concept, which includes data types and interfaces.

A complete discussion of when and how to use data types and interfaces structure diagrams is beyond the scope of this article. So why do I mentio here? There are times when you might want to model these classifier type is important to use the proper notation in doing so, or at least be aware of these classifiers incorrectly will likely confuse readers of your structure dia will probably not meet requirements.

A class and an interface differ: A class can have an actual instance of its t have at least one class to implement it. In UML 2, an interface is consider class modeling element. Therefore, an interface is drawn just like a class, rectangle also has the text "«interface»", as shown in Figure 10. [Note: W is completely within UML specification to put «class» in the top compartm would with «interface»; however, the UML specification says that placing t compartment is optional, and it should be assumed if «class» is not displa

Figure 10: Example of a class diagram in which the Professor and St Person interface



In the diagram shown in Figure 10, both the Professor and Student classe interface and do not inherit from it. We know this for two reasons: 1) The I interface — it has the "«interface»" text in the object's name area, and we Student objects are *class* objects because they are labeled according to t object (there is no additional classification text in their name area). 2) We shown here, because the line with the arrow is dotted and not solid. As sh with a closed, unfilled arrow means realization (or implementation); as we line with a closed, unfilled arrow means inheritance.

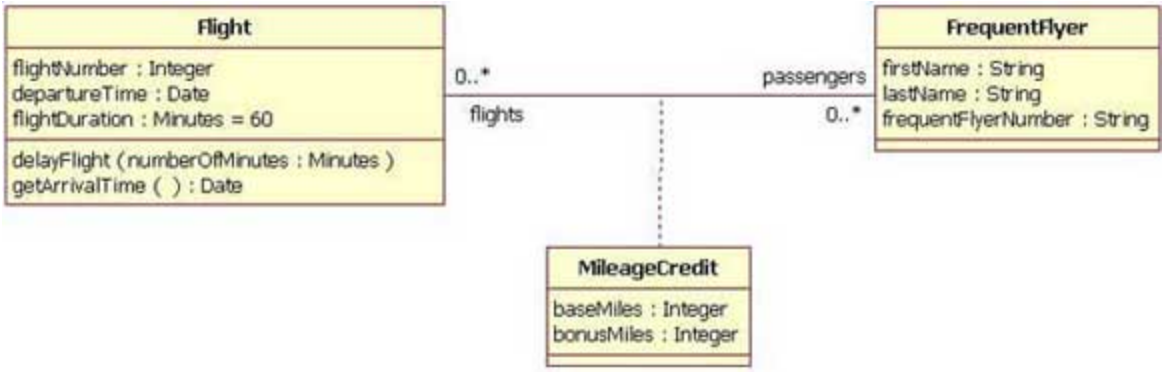
More associations

Above, I discussed bi-directional and uni-directional associations. Now I w types of associations.

Association class

In modeling an association, there are times when you need to include anc valuable information about the relationship. For this you would use an *ass* primary association. An association class is represented like a normal cla association line between the primary classes intersects a dotted line conn Figure 11 shows an association class for our airline industry example.

Figure 11: Adding the association class MileageCredit



In the class diagram shown in Figure 11, the association between the Flig class results in an association class called MileageCredit. This means tha class is associated with an instance of a FrequentFlyer class, there will al MileageCredit class.

Aggregation

Aggregation is a special type of association used to model a "whole to its aggregation relationships, the lifecycle of a *part* class is independent from

For example, we can think of *Car* as a whole entity and *Car Wheel* as par can be created weeks ahead of time, and it can sit in a warehouse before assembly. In this example, the Wheel class's instance clearly lives indepe instance. However, there are times when the *part* class's lifecycle *is not* in *whole* class — this is called composition aggregation. Consider, for exam company to its departments. Both *Company and Departments* are modele cannot exist before a company exists. Here the Department class's instan existence of the Company class's instance.

Let's explore basic aggregation and composition aggregation further.

Basic aggregation

An association with an aggregation relationship indicates that one class is aggregation relationship, the child class instance can outlive its parent cla relationship, you draw a solid line from the parent class to the part class, a

shape on the parent class's association end. Figure 12 shows an example between a Car and a Wheel.

Figure 12: Example of an aggregation association

Composition aggregation

The composition aggregation relationship is just another form of the aggre child class's instance lifecycle is dependent on the parent class's instance shows a composition relationship between a Company class and a Depar composition relationship is drawn like the aggregation relationship, but thi filled.

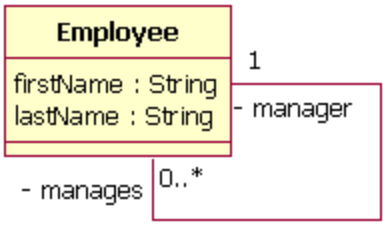
Figure 13: Example of a composition relationship

In the relationship modeled in Figure 13, a Company class instance will al Department class instance. Because the relationship is a composition rela instance is removed/destroyed, the Department instance is automatically Another important feature of composition aggregation is that the part class instance of the parent class (e.g. the Company class in our example).

Reflexive associations

We have now discussed all the association types. As you may have notic shown a relationship between two different classes. However, a class can using a reflexive association. This may not make sense at first, but remem abstractions. Figure 14 shows how an Employee class could be related to manager/manages role. When a class is associated to itself, this does not related to itself, but that an instance of the class is related to another insta

Figure 14: Example of a reflexive association relationship



The relationship drawn in Figure 14 means that an instance of Employee Employee instance. However, because the relationship role of "manages" Employee might not have any other Employees to manage.

Visibility

In object-oriented design, there is a notation of visibility for attributes and types of visibility: public, protected, private, and package.

The UML specification does not require attributes and operations visibility diagram, but it does require that it be defined for each attribute or operatic class diagram, you place the visibility mark in front of the attribute's or ope specifies four visibility types, an actual programming language may add a not support the UML-defined visibilities. Table 4 displays the different mar visibility types.

Table 4: Marks for UML-supported visibility types

Mark	Visibility type
+	Public
#	Protected
-	Private
~	Package

Now, let's look at a class that shows the visibility types indicated for its att Figure 15, all the attributes and operations are public, with the exception The updateBalance operation is protected.

Figure 15: A BankAccount class that shows the visibility of its attrib

UML 2 additions

Now that we have covered the basics and the advanced topics, we will co
added to the class diagram from UML 1.x.

Instances

When modeling a system's structure it is sometimes useful to show exam
model this, UML 2 provides the *instance specification* element, which sho
example (or real) instances in the system.

The notation of an instance is the same as a class, but instead of the top
class's name, the name is an underlined concatenation of:

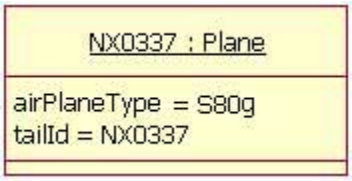
```
Instance Name : Class Name
```

For example:

```
Donald : Person
```

Because the purpose of showing instances is to show interesting or releva
necessary to include in your model the entire instance's attributes and op
appropriate to show only the attributes and their values that are interestin

Figure 16: An example instance of a Plane class (only the interesting



However, merely showing some instances without their relationship is not
allows for the modeling of the relationships/associations at the instance le

drawing associations are the same as for normal class relationships, altho
requirement when modeling the associations. The additional restriction is
must match the class diagram's relationships and therefore the associatio
the class diagram. An example of this is shown in Figure 17. In this exam
instances of the class diagram found in Figure 6.

Figure 17: An example of Figure 6 using instances instead of classes

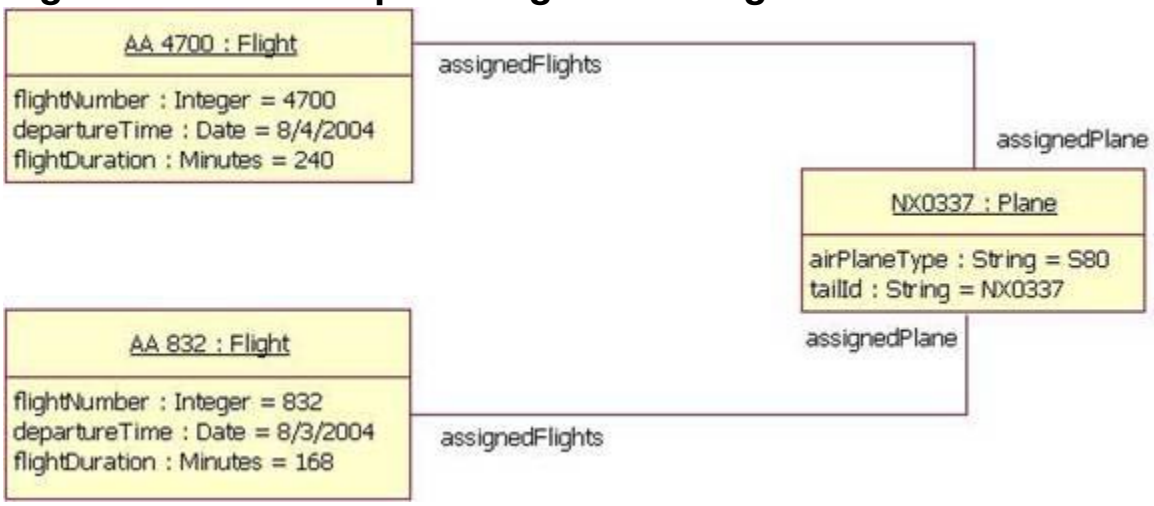


Figure 17 has two instances of the Flight class because the class diagram
between the Plane class and the Flight class is *zero-to-many*. Therefore,
Flight instances that the NX0337 Plane instance is related to.

Roles

Modeling the instances of classes is sometimes more detailed than one m
simply want to model a class's relationship at a more generic level. In suc
role notation. The role notation is very similar to the instances notation. To
a box and place the class's role name and class name inside as with the i
case you do not underline the words. Figure 18 shows an example of the
class described by the diagram at Figure 14. In Figure 18, we can tell, eve
related to itself, that the relationship is really between an Employee playin
Employee playing the role of team member.

Figure 18: A class diagram showing the class in Figure 14 in its diffe

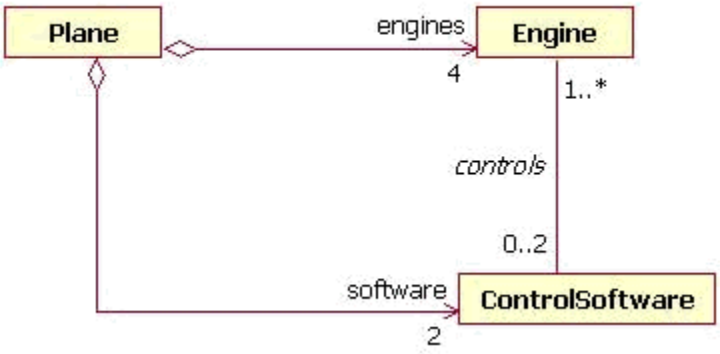
Note that you cannot model a class's role on a plain class diagram, even t
appear that you can. In order to use the role notation you will need to use
discussed next.

Internal Structures

One of the more useful features of UML 2 structure diagrams is the new i
allows you to show how a class or another classifier is internally compose
1.x, because the notation set limited you to showing only the aggregation
Now, in UML 2, the internal structure notation lets you more clearly show
each other.

Let's look at an example. In Figure 18 we have a class diagram showing h
of four engines and two control software objects. What is missing from thi
about how airplane parts are assembled. From the diagram in Figure 18,
software objects control two engines each, or if one control software obje
the other controls one engine.

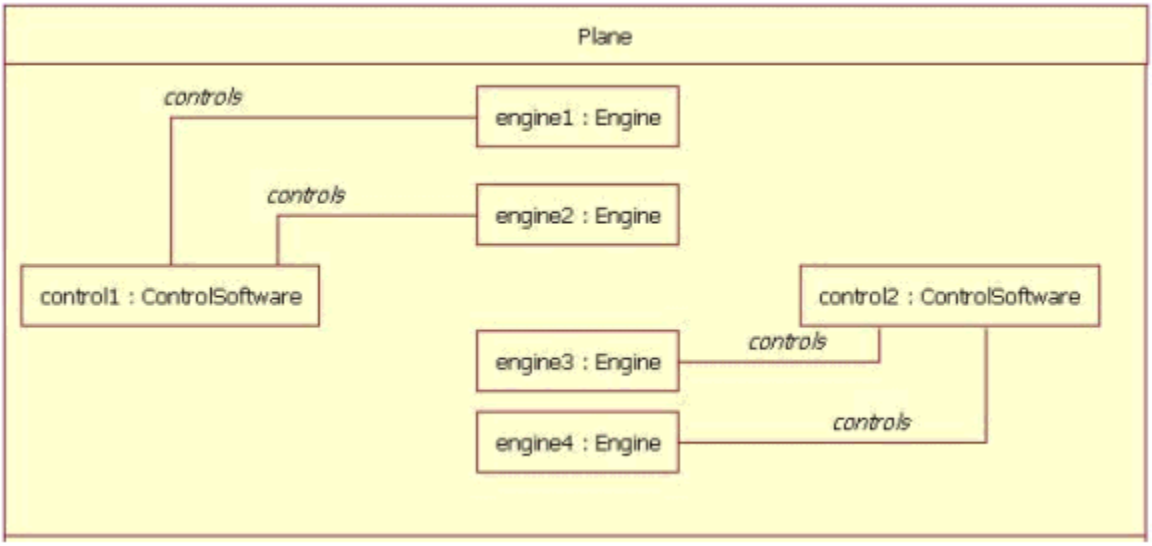
Figure 19: A class diagram that only shows relationships between th



Drawing a class's internal structure will improve this situation. You start by
compartments. The top compartment contains the class name, and the lo
class's internal structure, showing the parent class's part classes in their r

each particular class relates to others in that role. Figure 19 shows the int
notice how the internal structure clears up the confusion.

Figure 20: An example internal structure of a Plane class



In Figure 20 the Plane has two ControlSoftware objects and each one cor
ControlSoftware on the left side of the diagram (control1) controls engines
on the right side of the diagram (control2) controls engines 3 and 4.

Conclusion

There are at least two important reasons for understanding the class diag
the static structure of classifiers in a system; the second reason is that the
notation for other structure diagrams prescribed by UML. Developers will
created specially for them; but other team members will find them useful,
class diagrams to model systems from the business perspective. As we w
series on UML basics, other diagrams — including the activity, sequence,
refer to the classes modeled and documented on the class diagram.

Next in this series on UML basics: The component diagram.

Resources

D
o

Learn

Understand more about how the new functionality of the UML Modeler component common to both IBM Rational Software Architect Standard Edition Version 7.5 and IBM Rational Software Architect for WebSphere Software Version 7.5 by reading this article Using the new features of UML Modeler in IBM Rational Software Architect Version 7.5.

Understand more about the new functionality of Rational Software Architect for WebSphere Software Version 7.5 by reading this article Overview of Rational Software Architect for WebSphere Software Version 7.5.

Learn about Rational Tau.

Learn about other applications in the IBM Rational Software Delivery Platform, including collaboration tools for parallel development and geographically dispersed teams, plus specialized software for architecture management, asset management, change and release management, integrated requirements management, process and portfolio management, and quality management. You can find product manuals, installation guides, and other documentation in the IBM Rational Online Documentation Center.

Visit the Rational software area on developerWorks for technical resources and best practices for Rational Software Delivery Platform products.

Explore Rational computer-based, Web-based, and instructor-led online courses. Hone your skills and learn more about Rational tools with these courses, which range from introductory to advanced. The courses on this catalog are available for purchase through computer-based training or Web-based training. Some of the "Getting Started" courses are available free of charge.

Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.

Browse the technology bookstore for books on these and other technical

topics.

Get products and technologies

Download a trial version of Rational Software Modeler.

Download a trial version of Rational Software Architect standard edition.

Download Rational Software Architect for Websphere Software.

Download Other IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

Check out developerWorks blogs and get involved in the developerWorks community.

Join the Development Tools forum on developerWorks to discuss Rational Application Developer, Rational Software Architect, and Rational Software Modeler.

Join the Rational Tau forum on developerWorks.