WEEK- 1

Write Problem Statement for Project on Class Diagram

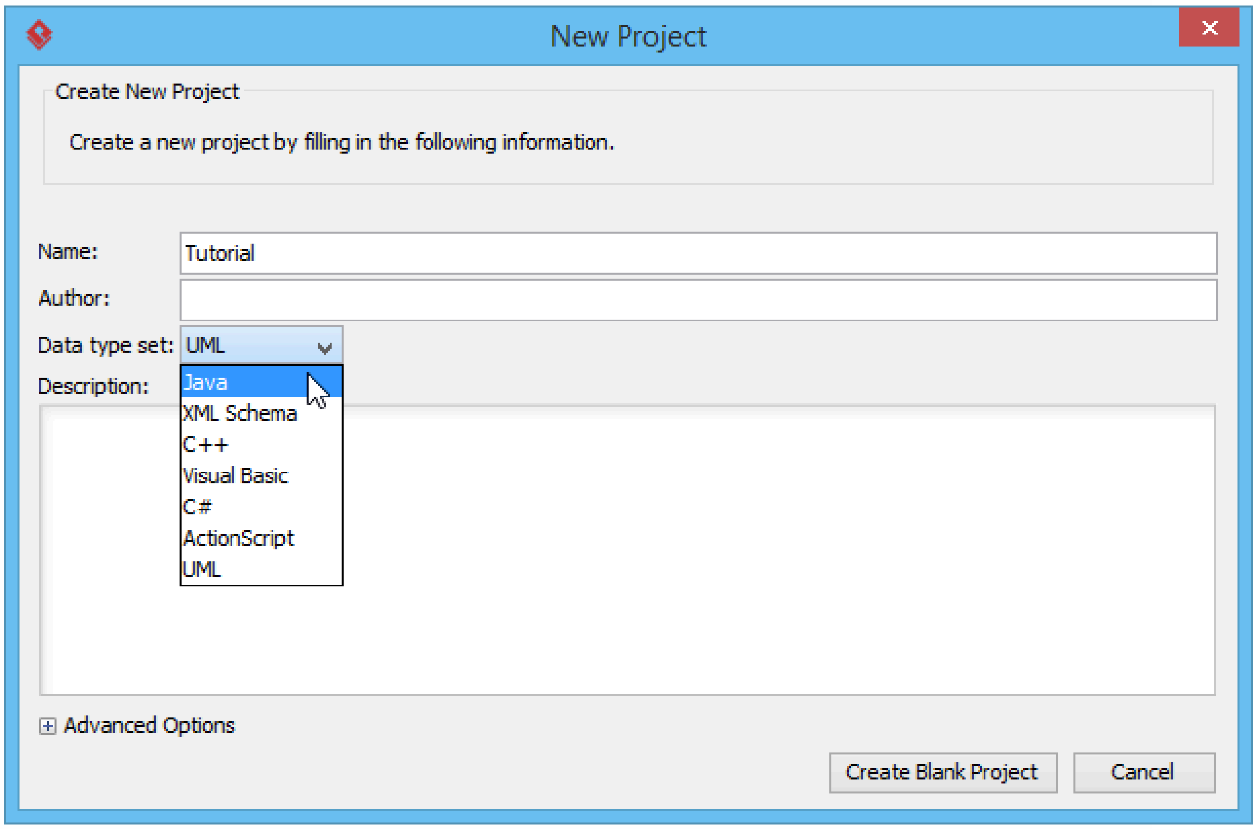
PROBLEM STATEMENTS:

1. Create a detailed UML class diagram representing the ATM system, including all necessary classes, attributes, and methods.
2. To create the relationships and interactions between different classes, showing associations, generalizations, and dependencies.
3. To create a UML class diagram for an Automated Teller Machine (ATM) system, demonstrating the relationships and interactions between different classes involved in the system's operations.

Creating a class Diagram

1. Select **Project > New** from the application toolbar.
2. In the **New Project** window, enter *Tutorial* as Name.
3.
4. By default, **UML** is selected to be the **Data type set**, meaning that you can use the primitive UML data types when constructing your model. Let's say we are going to draw a class diagram for a Java project. Select **Java**

to be the **Data type set**.


New Project

Create New Project

Create a new project by filling in the following information.

Name: Tutorial

Author:

Data type set: UML

Description:

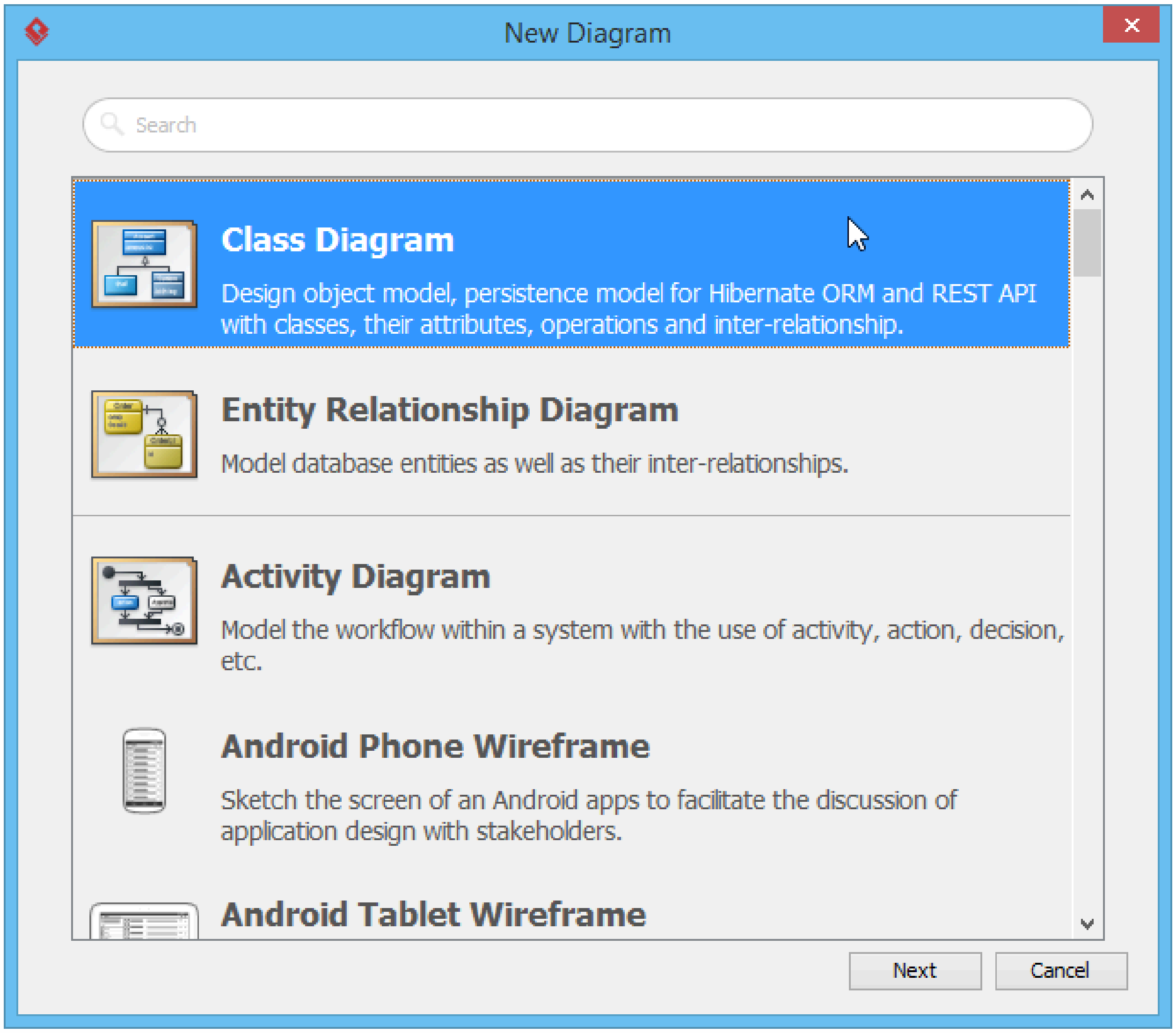
5. Click **Create Blank Project**.

Creating a simple UML class diagram

In this section you will create a class diagram with one class, and several attributes in it. You will be creating the attributes with primitive Java data types.

1. Create a UML class diagram first. You can create a class diagram by selecting **Diagram** > **New** from the application toolbar. Select **Class Diagram** in the **New Diagram** window and then click **Next**. Click **OK** again to

create the diagram.


New Diagram

Search

Class Diagram
Design object model, persistence model for Hibernate ORM and REST API with classes, their attributes, operations and inter-relationship.

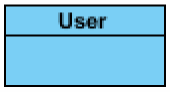
Entity Relationship Diagram
Model database entities as well as their inter-relationships.

Activity Diagram
Model the workflow within a system with the use of activity, action, decision, etc.

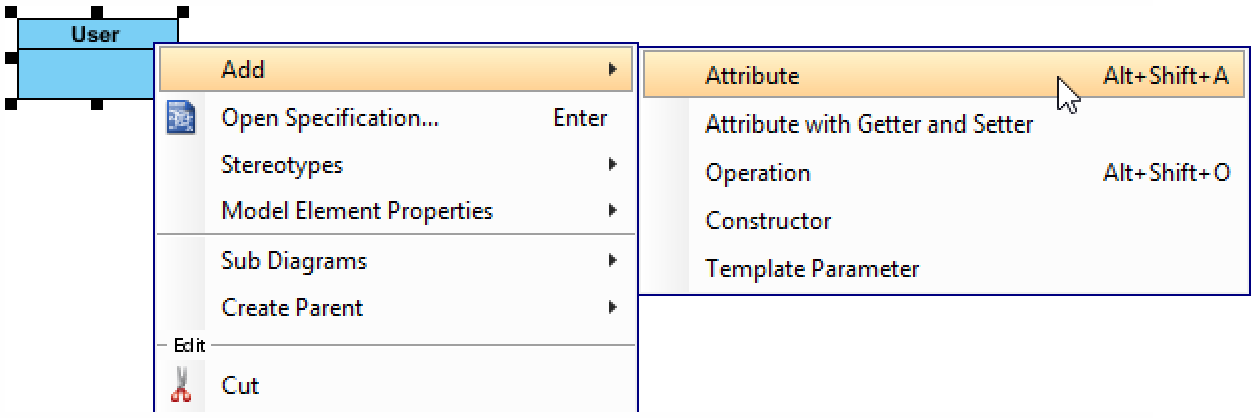
Android Phone Wireframe
Sketch the screen of an Android apps to facilitate the discussion of application design with stakeholders.

Android Tablet Wireframe

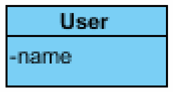
2. Click a class *User*.



3. Let's add an attribute name into the class. Right-click on the class and select **Add > Attribute** from the popup menu.

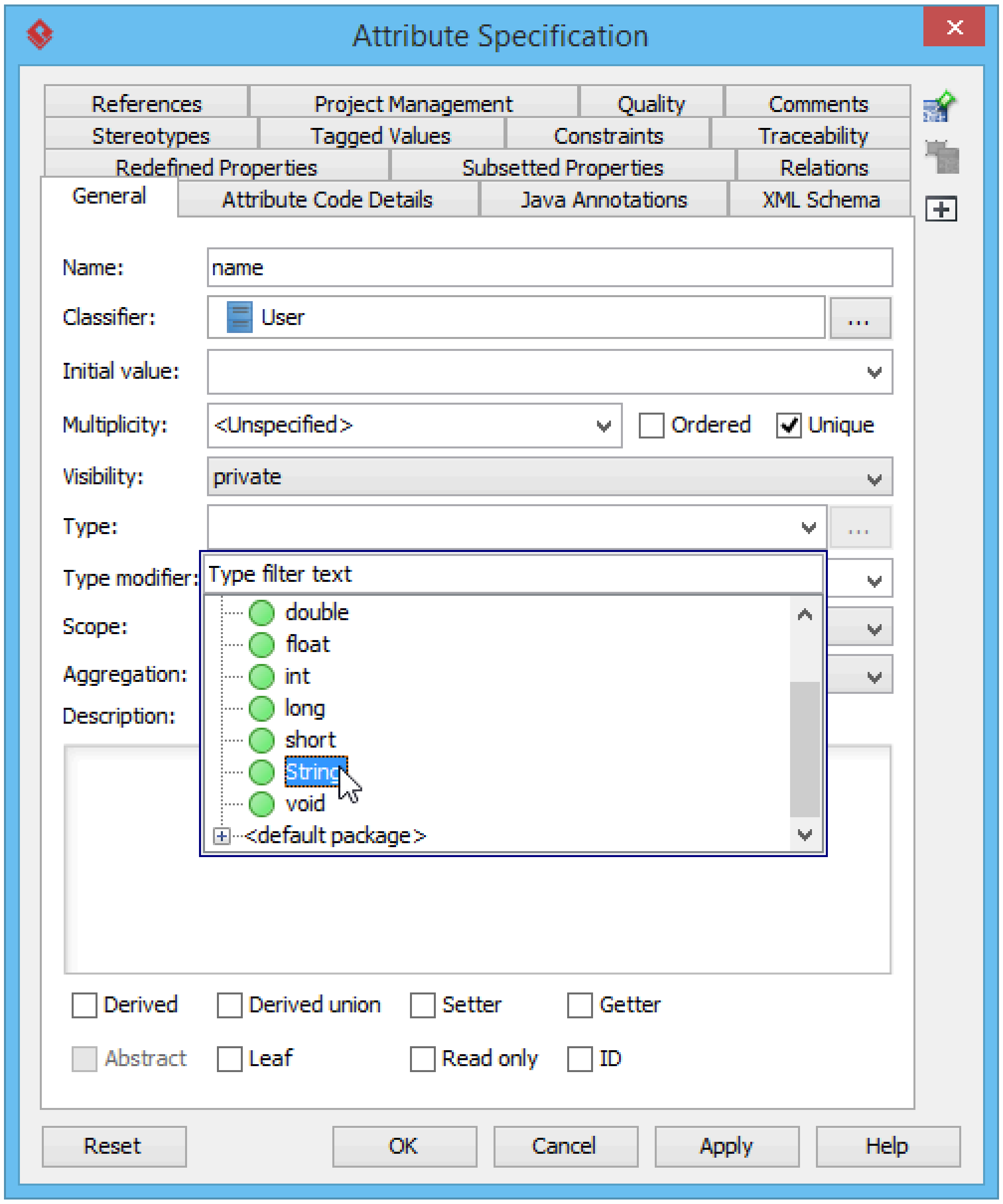


4. *name* is a (Java) String attribute. You can type *name : String* to create such an attribute but let's try something different this time. Type *name* and then click on the diagram background to create a typeless attribute.



5. Right-click on the attribute and select **Open Specification...** from the popup menu.
6. Click on the drop down menu next to the **Type** field. You can see a list of primitive Java data types available for selection. Now, select **String**

and click **OK** to confirm.



Name: name

Classifier: User

Initial value:

Multiplicity: <Unspecified> ☐ Ordered ☑ Unique

Visibility: private

Type:

Type modifier:

Scope:

Aggregation:

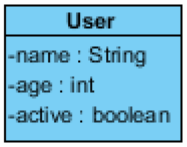
Description:

Type filter text
- double
- float
- int
- long
- short
- String
- void
- <default package>

☐ Derived ☐ Derived union ☐ Setter ☐ Getter

☐ Abstract ☐ Leaf ☐ Read only ☐ ID

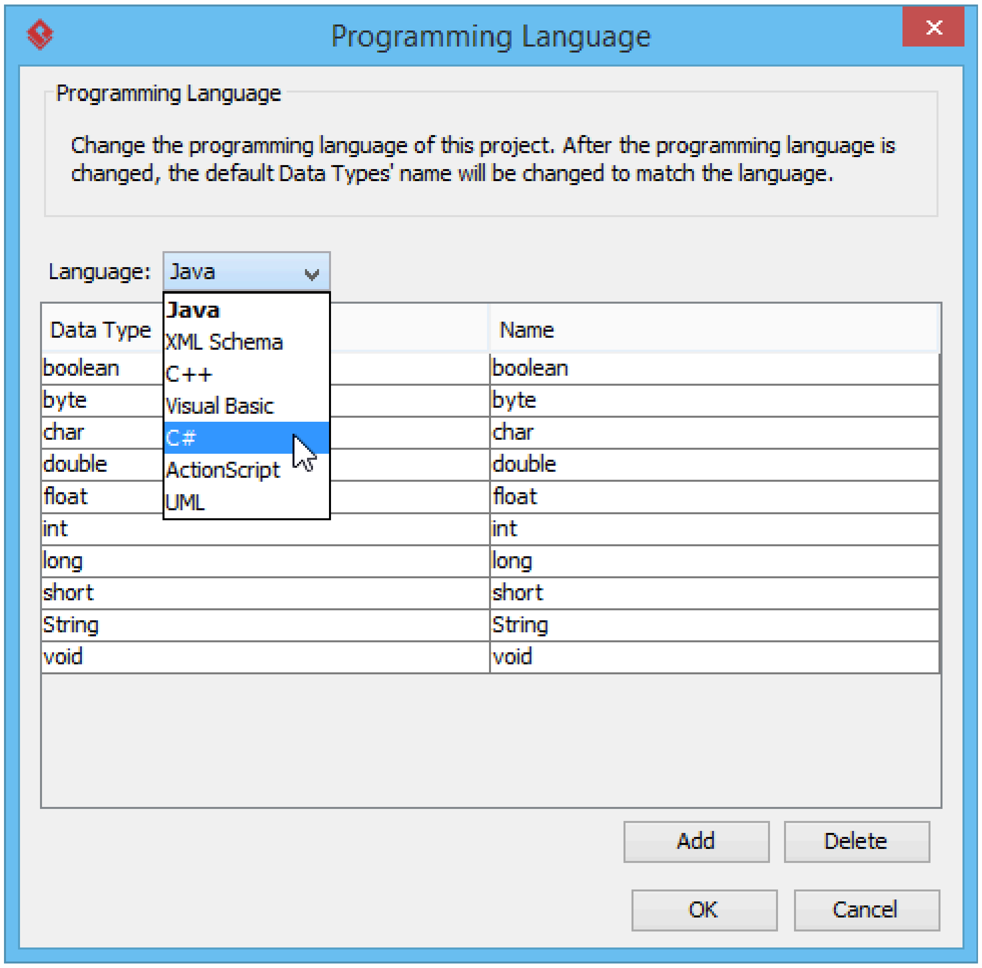
7. Now, create two more attributes *age : int* and *active : boolean*. To save time, you can type the name and data type inline without going through the specification window.



Presenting class model in another programming language

Now you have a class diagram with Java data types used as attribute types. Your Java developers are happy. Let's entertain the C# developers by presenting the data types in C#.

1. Select **Window** > **Configuration** > **Configure Programming Language** from the application toolbar.
2. The **Programming Language** window shows the currently selected language, its supported data types and their corresponding display names. We will describe in more detail in a minute. Now, change **Language** from **Java** to **C#**.



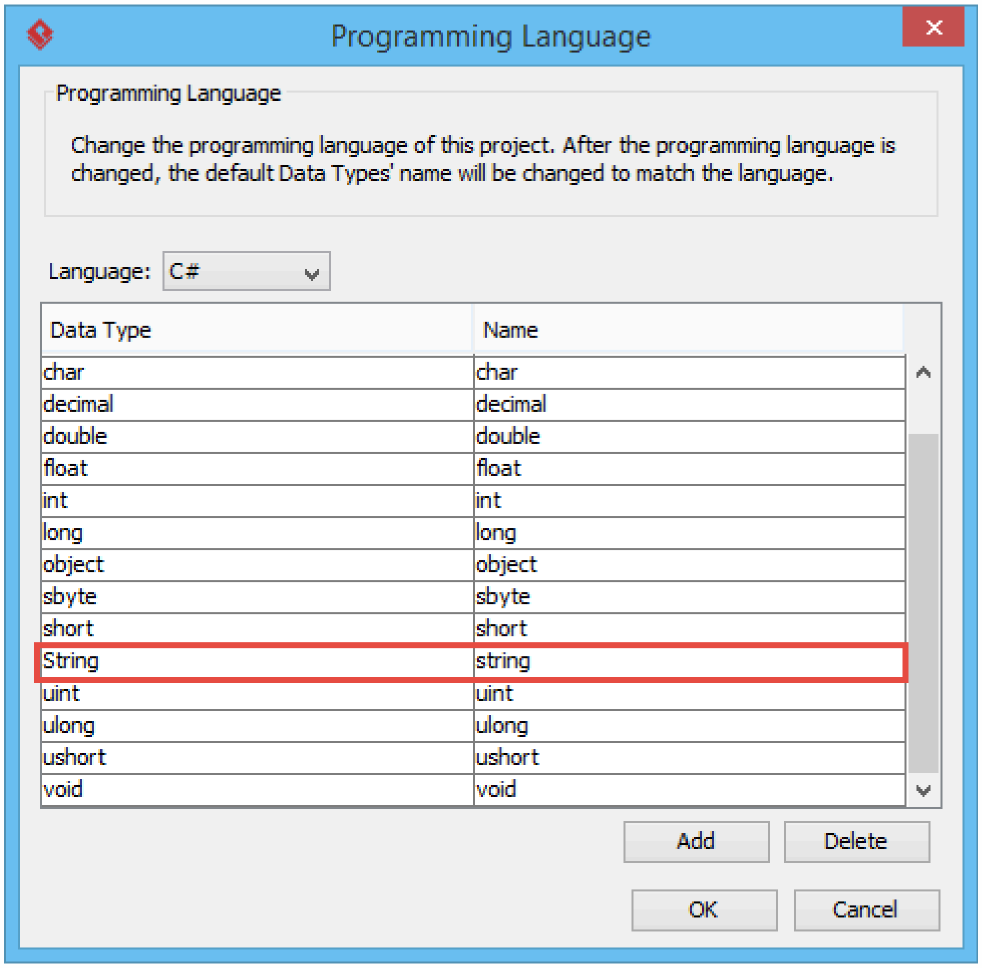
Change the programming language of this project. After the programming language is changed, the default Data Types' name will be changed to match the language.

Language: Java

Java
XML Schema
C++
Visual Basic
C#
ActionScript
UML

Data Type	Name
boolean	boolean
byte	byte
char	char
double	double
float	float
int	int
long	long
short	short
String	String
void	void

The list of data types is updated, and is now longer than before. If you scroll you can see some C# types like uint and ulong, which are not available in Java. So how to read the two columns? Let's check the row for String type. The first and second columns are showing String and string respectively. This means that the original String type (available under Java) will be displayed as string by changing the language to C#.



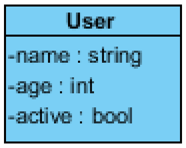
Programming Language

Change the programming language of this project. After the programming language is changed, the default Data Types' name will be changed to match the language.

Language: C#

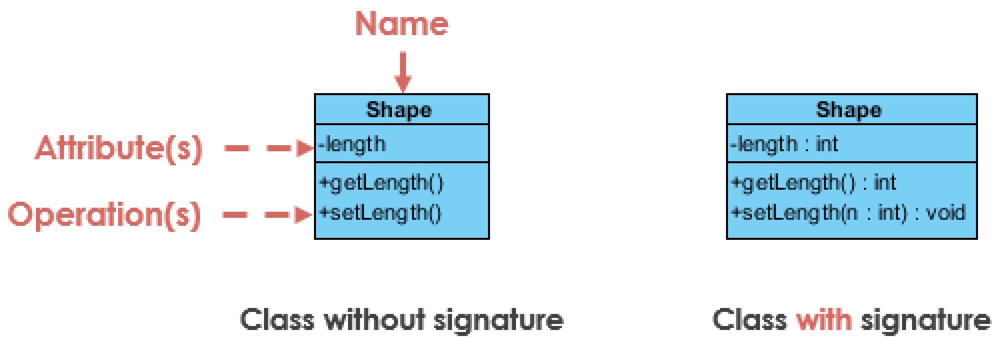
Data Type	Name
char	char
decimal	decimal
double	double
float	float
int	int
long	long
object	object
sbyte	sbyte
short	short
String	string
uint	uint
ulong	ulong
ushort	ushort
void	void

3. Click **OK** to confirm the change of programming language. You can now see the attributes *name* is now showing as a C# string, while *active* is now a C# bool instead of a Java boolean.



UML Class Notation

A class represent a concept which encapsulates state (attributes) and behavior (operations). Each attribute has a type. Each operation has a signature. *The class name is the only mandatory information.*



Class Name:

- The name of the class appears in the first partition.

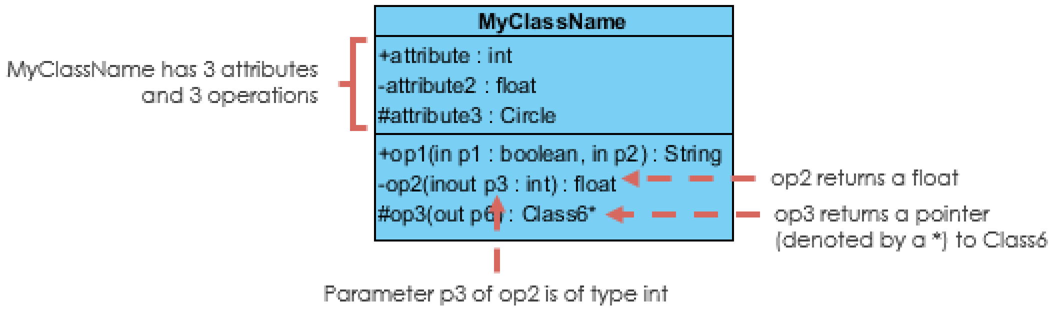
Class Attributes:

- Attributes are shown in the second partition.
- The attribute type is shown after the colon.
- Attributes map onto member variables (data members) in code.

Class Operations (Methods):

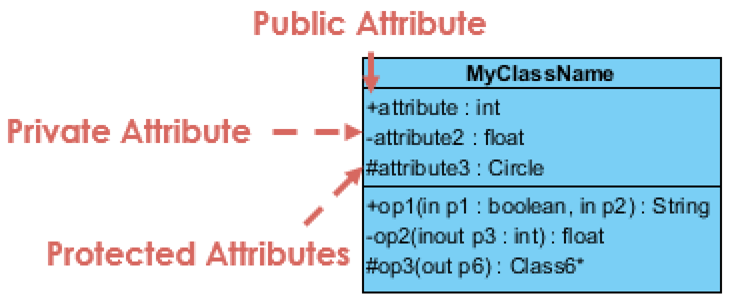
- Operations are shown in the third partition. They are services the class provides.

- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters are shown after the colon following the parameter name. Operations map onto class methods in code


MyClassName
+attribute : int
-attribute2 : float
#attribute3 : Circle
+op1(in p1 : boolean, in p2) : String
-op2(inout p3 : int) : float
#op3(out p6) : Class6*
MyClassName has 3 attributes and 3 operations
op2 returns a float
op3 returns a pointer (denoted by a *) to Class6
Parameter p3 of op2 is of type int

Class Visibility

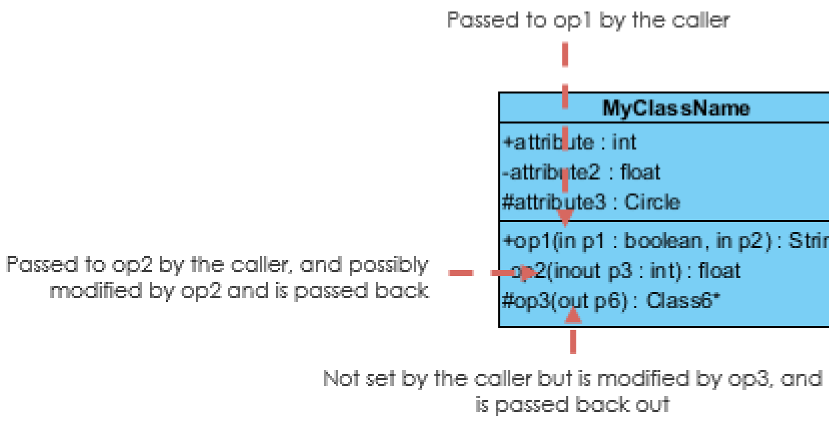
The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.



- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

Parameter Directionality

Each parameter in an operation (method) may be denoted as in, out or inout which specifies its direction with respect to the caller. This directionality is shown before the parameter name.



Passed to op1 by the caller
MyClassName
+attribute : int
-attribute2 : float
#attribute3 : Circle
+op1(in p1 : boolean, in p2) : String
-op2(inout p3 : int) : float
#op3(out p6) : Class6*
Passed to op2 by the caller, and possibly modified by op2 and is passed back
Not set by the caller but is modified by op3, and is passed back out

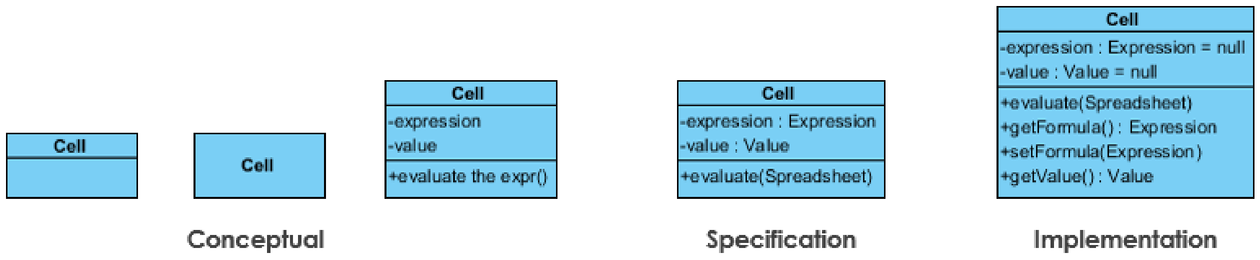
Perspectives of Class Diagram

The choice of perspective depends on how far along you are in the development process. During the formulation of a domain model, for example, you would seldom move past the conceptual perspective. Analysis models will typically feature a mix of conceptual and specification perspectives. Design model development will typically start with heavy emphasis on the specification perspective, and evolve into the implementation perspective.

A diagram can be interpreted from various perspectives:

- Conceptual: represents the concepts in the domain
- Specification: focus is on the interfaces of Abstract Data Type (ADTs) in the software
- Implementation: describes how classes will implement their interfaces

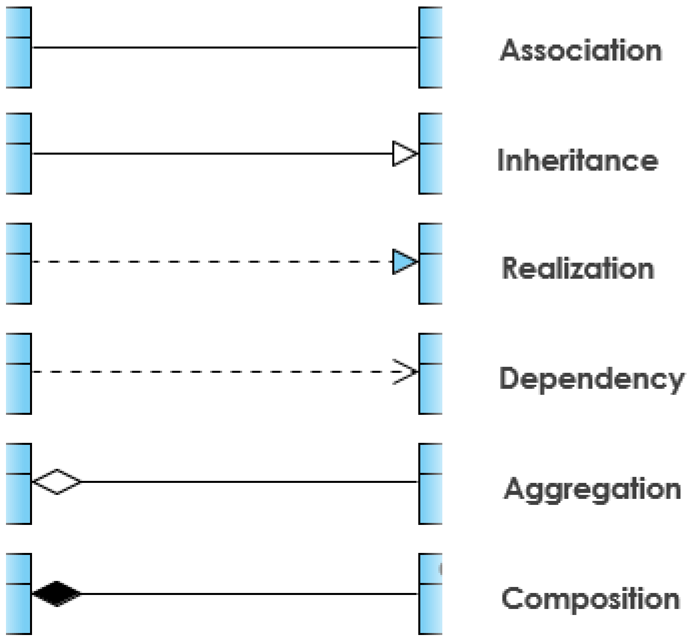
The perspective affects the amount of detail to be supplied and the kinds of relationships worth presenting. As we mentioned above, the class name is the only mandatory information.



Relationships between classes

UML is not just about pretty pictures. If used correctly, UML precisely conveys how code should be implemented from diagrams. If precisely interpreted, the implemented code will correctly reflect the intent of the designer. Can you describe what each of the relationships mean relative to your target programming language shown in the Figure below?

If you can’t yet recognize them, no problem this section is meant to help you to understand UML class relationships. A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:

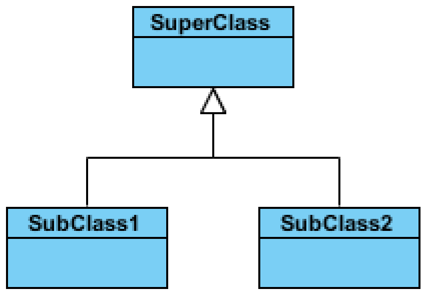


Inheritance (or Generalization):

A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.

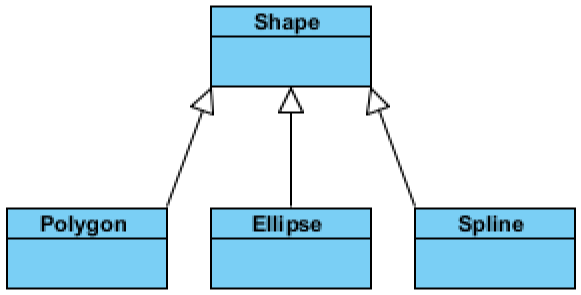
- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of SuperClass.

The figure below shows an example of inheritance hierarchy. SubClass1 and SubClass2 are derived from SuperClass. The relationship is displayed as a solid line with a hollow arrowhead that points from the child element to the parent element.

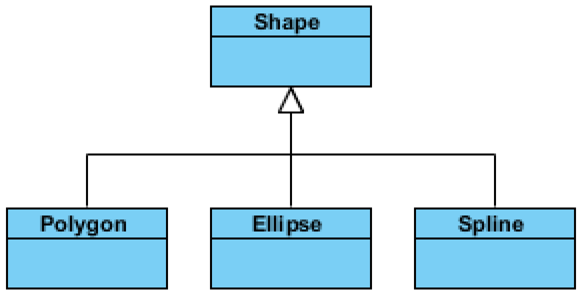


Inheritance Example - Shapes

The figure below shows an inheritance example with two styles. Although the connectors are drawn differently, they are semantically equivalent.



Style 1: Separate target



Style 2: Shared target

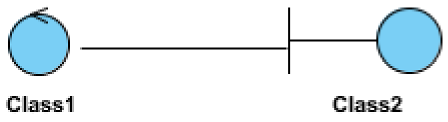
Association

Associations are relationships between classes in a UML Class Diagram. They are represented by a solid line between classes. Associations are typically named using a verb or verb phrase which reflects the real world problem domain.

Simple Association

- A structural link between two peer classes.
- There is an association between Class1 and Class2

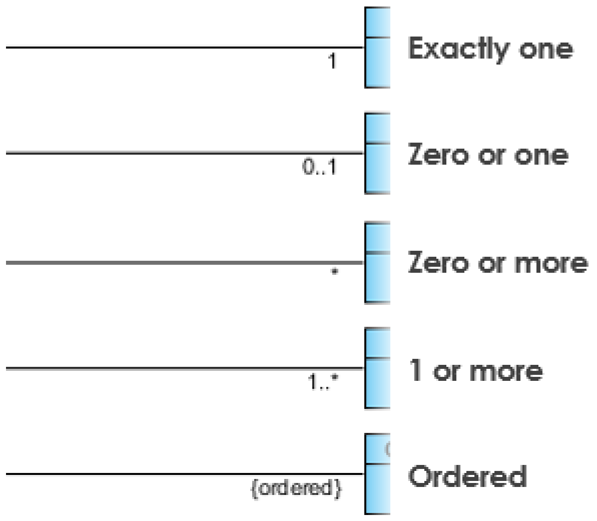
The figure below shows an example of simple association. There is an association that connects the <<control>> class Class1 and <<boundary>> class Class2. The relationship is displayed as a solid line connecting the two classes.



Cardinality

Cardinality is expressed in terms of:

- one to one
- one to many
- many to many

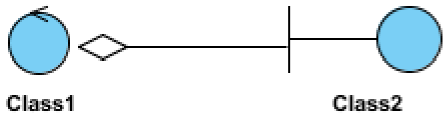


Aggregation

A special type of association.

- It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.

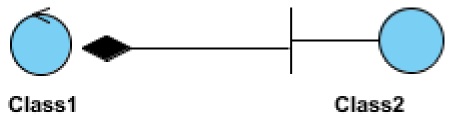
The figure below shows an example of aggregation. The relationship is displayed as a solid line with a unfilled diamond at the association end, which is connected to the class that represents the aggregate.



Composition

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.

The figure below shows an example of composition. The relationship is displayed as a solid line with a filled diamond at the association end, which is connected to the class that represents the whole or composite.

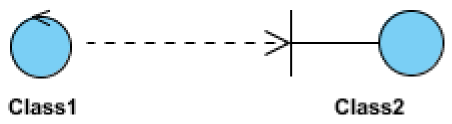


Dependency

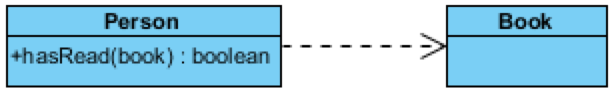
An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modeled as a dependency relationship.

- A special type of association.
- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2

The figure below shows an example of dependency. The relationship is displayed as a dashed line with an open arrow.



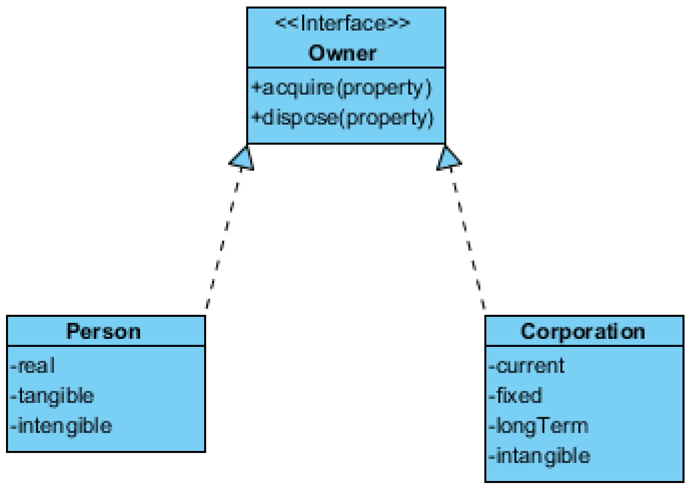
The figure below shows another example of dependency. The Person class might have a hasRead method with a Book parameter that returns true if the person has read the book (perhaps by checking some database).



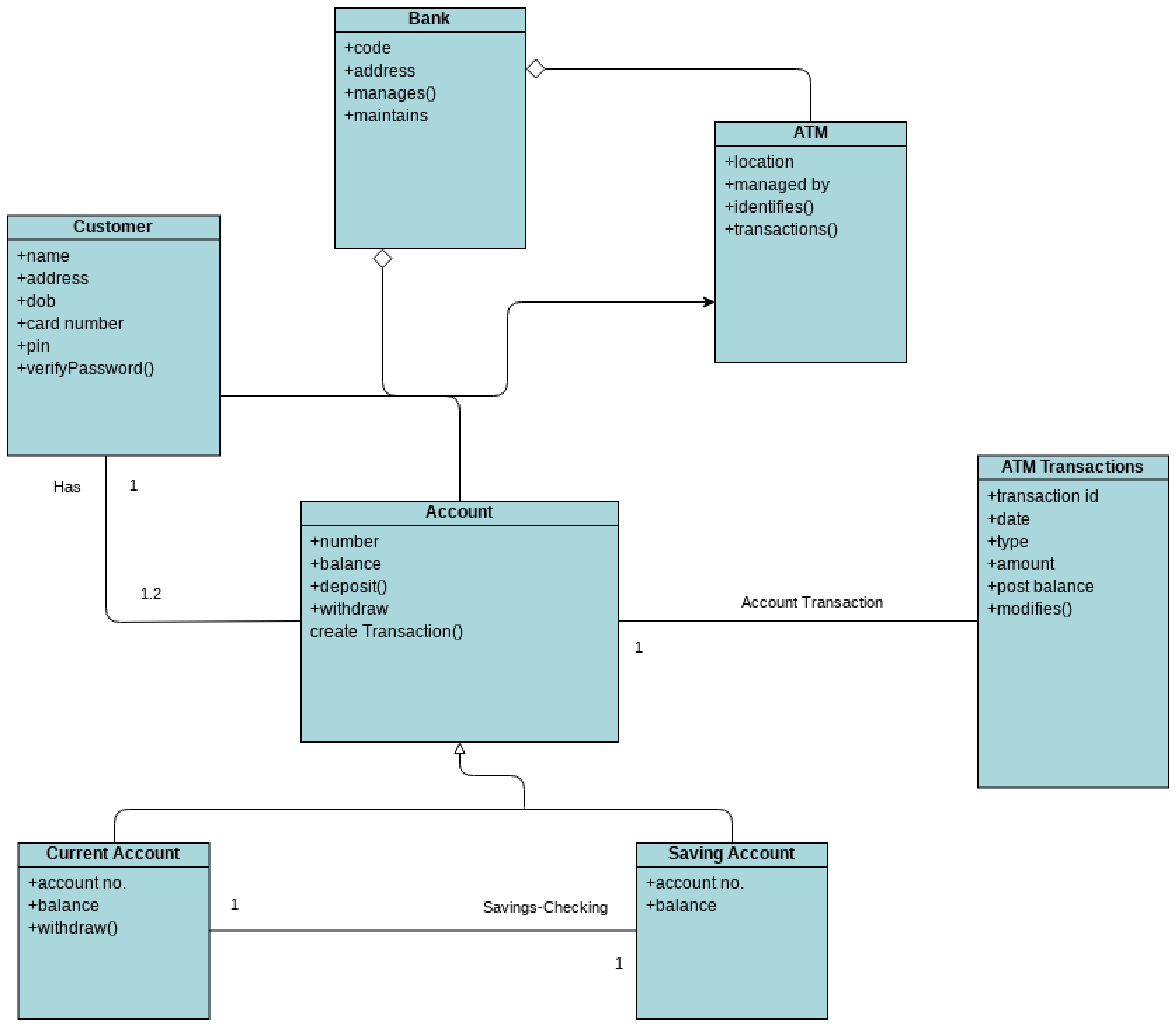
Realization

Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

For example, the Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.



Class Diagram Example: Order System


Bank
+code
+address
+manages()
+maintains
ATM
+location
+managed by
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1.2
Account
+number
+balance
+deposit()
+withdraw
create Transaction()
Account Transaction
1
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance