OO Analysis and Design with UML 2 and UP

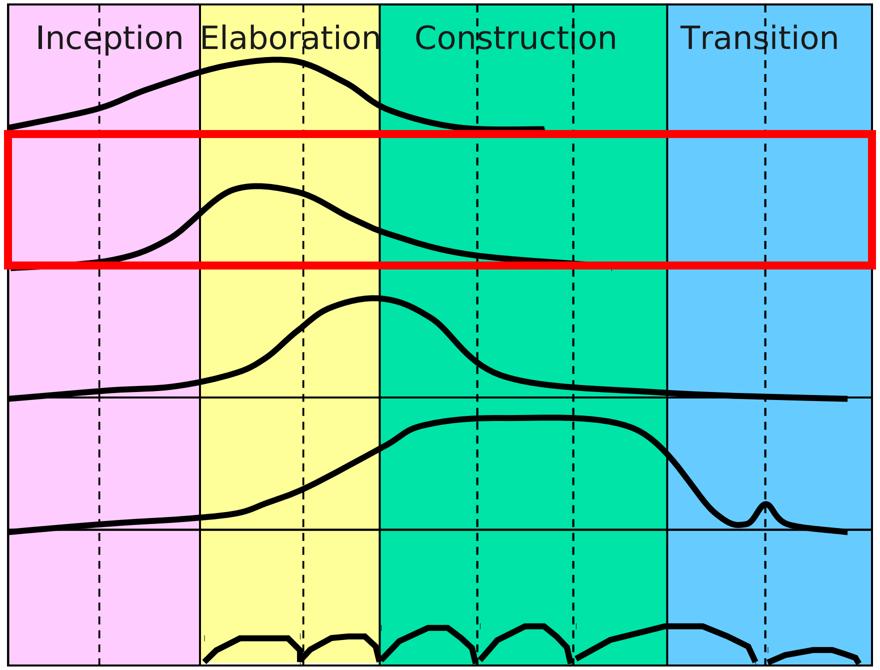
Dr. Jim Arlow,

Zuhlke Engineering Limited

Analysis - introduction

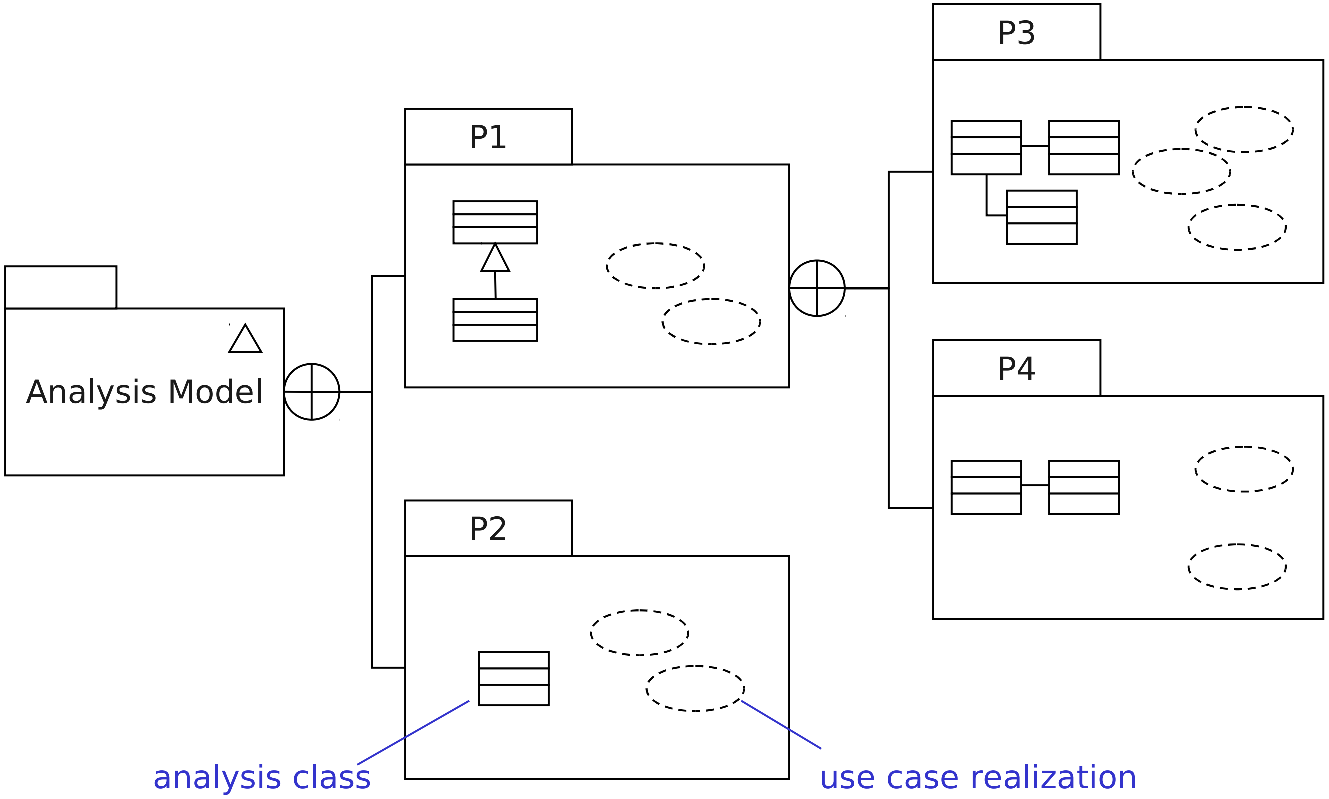
Analysis - purpose

- Produce an Analysis Model of the system’s desired behaviour:
  - This model should be a statement of what the system does not how it does it
  - We can think of the analysis model as a “first-cut” or “high level” design model
  - It is in the language of the business
- In the Analysis Model we identify:
  - Analysis classes
  - Use-case realizations



Analysis - metamodel

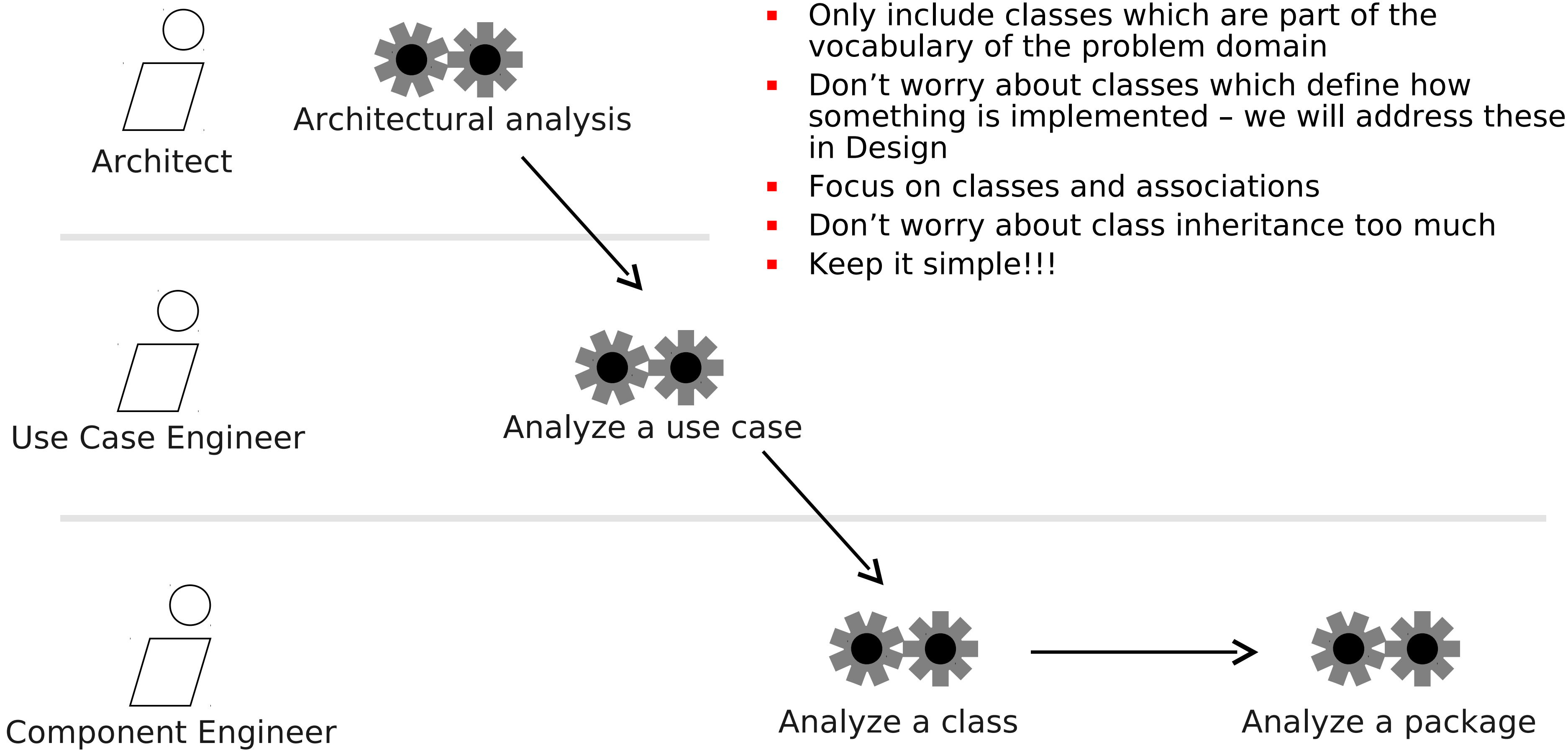
- Packages contain UML modelling elements and diagrams (we only show the elements here)
- Each element or diagram is owned by exactly one package



6.4

Workflow - Analysis

- Analysis guidelines: 6.5
  - 50 to 100 classes in the analysis model of a moderately complex system
  - Only include classes which are part of the vocabulary of the problem domain
  - Don’t worry about classes which define how something is implemented – we will address these in Design
  - Focus on classes and associations
  - Don’t worry about class inheritance too much
  - Keep it simple!!!



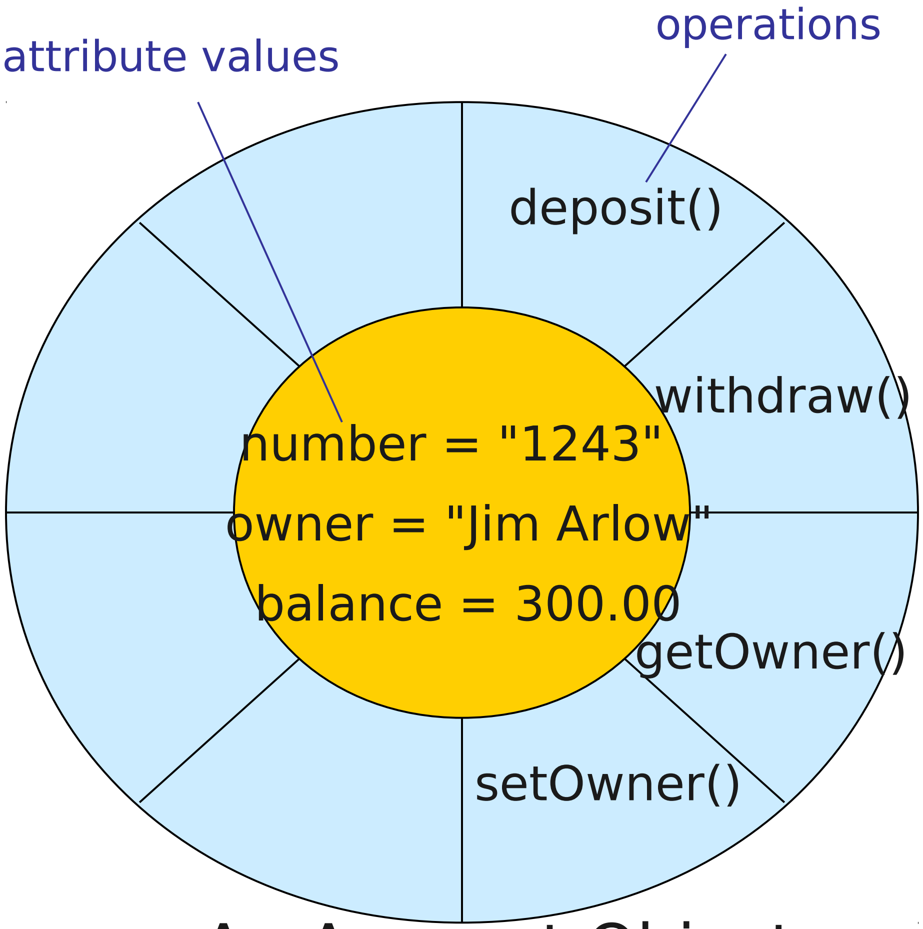
Analysis - objects and classes

What are objects?

- Objects consist of data and function packaged together in a reusable unit. Objects *encapsulate* data
- Every object is an instance of some *class* which defines the common set of *features* (attributes and operations) shared by all of its instances. Objects have:
  - Attribute values – the data part
  - Operations – the behaviour part
- All objects have:
  - *Identity*: Each object has its own unique identity and can be accessed by a unique handle
  - *State*: This is the actual data values stored in an object at any point in time
  - *Behaviour*: The set of operations that an object can perform

Encapsulation

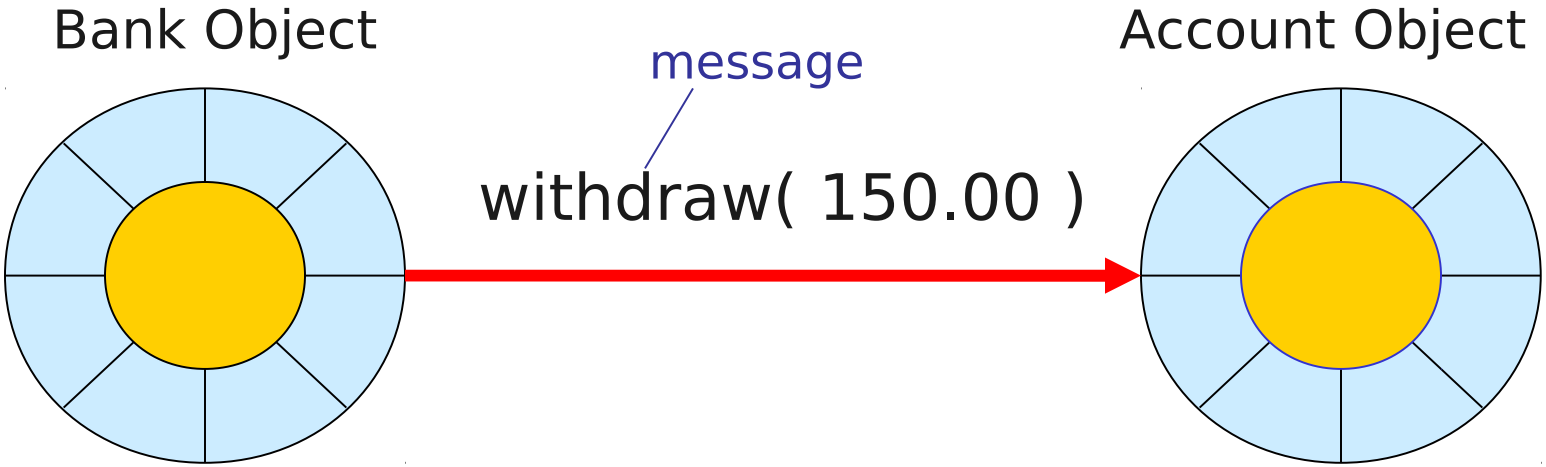
- Data is hidden inside the object. The only way to access the data is via one of the operations
- This is *encapsulation* or *data hiding* and it is a very powerful idea. It leads to more robust software and reusable code.



An Account Object

Messaging

- In OO systems, objects send messages to each other over links
- These messages cause an object to invoke an operation



Bank Object

message

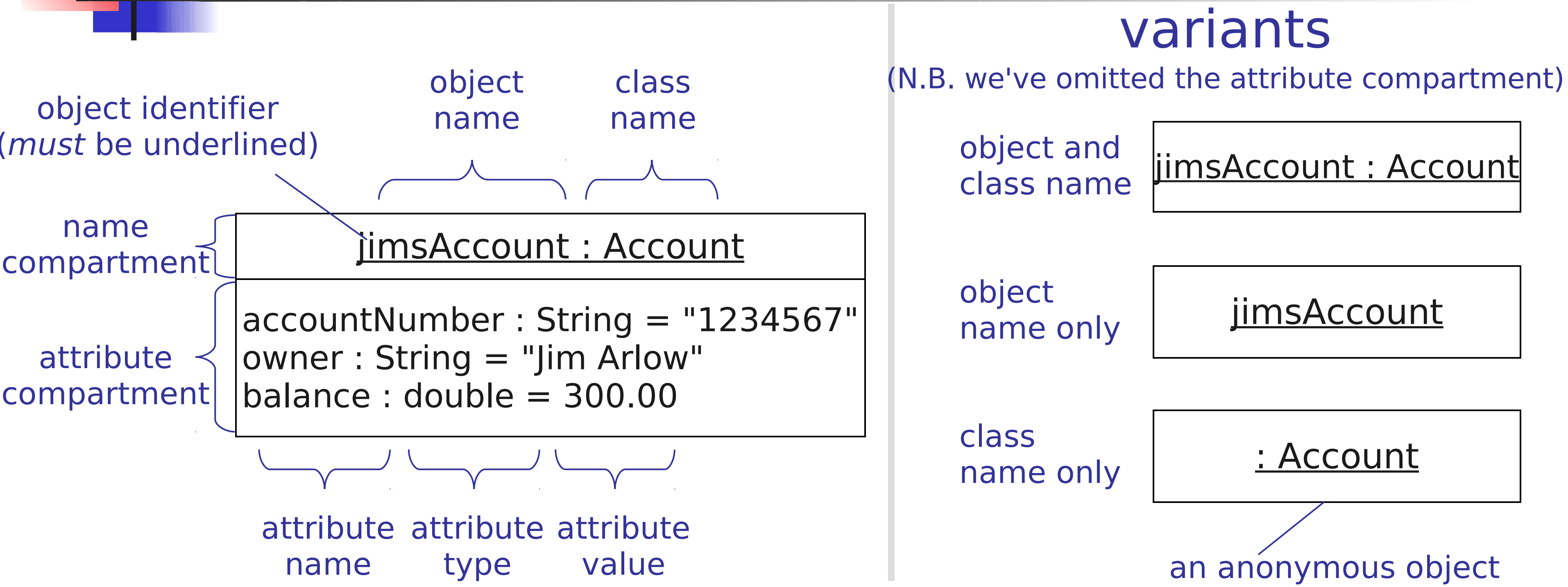
withdraw(150.00)

Account Object

the Bank object sends the message “withdraw 150.00” to an Account object.

the Account object responds by invoking its withdraw operation. This operation decrements the account balance by 150.00.

UML Object Syntax



object identifier
(*must* be underlined)

object name

class name

name compartment

attribute compartment

jimsAccount : Account

accountNumber : String = "1234567"
owner : String = "Jim Arlow"
balance : double = 300.00

attribute name

attribute type

attribute value

variants

(N.B. we've omitted the attribute compartment)

object and class name — jimsAccount : Account

object name only — jimsAccount

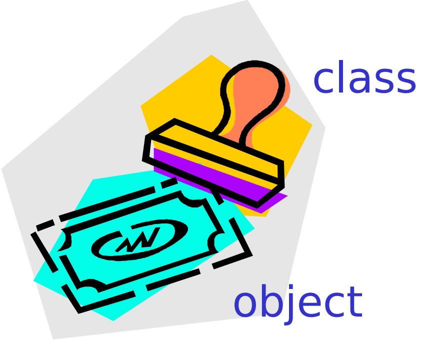
class name only — : Account

an anonymous object

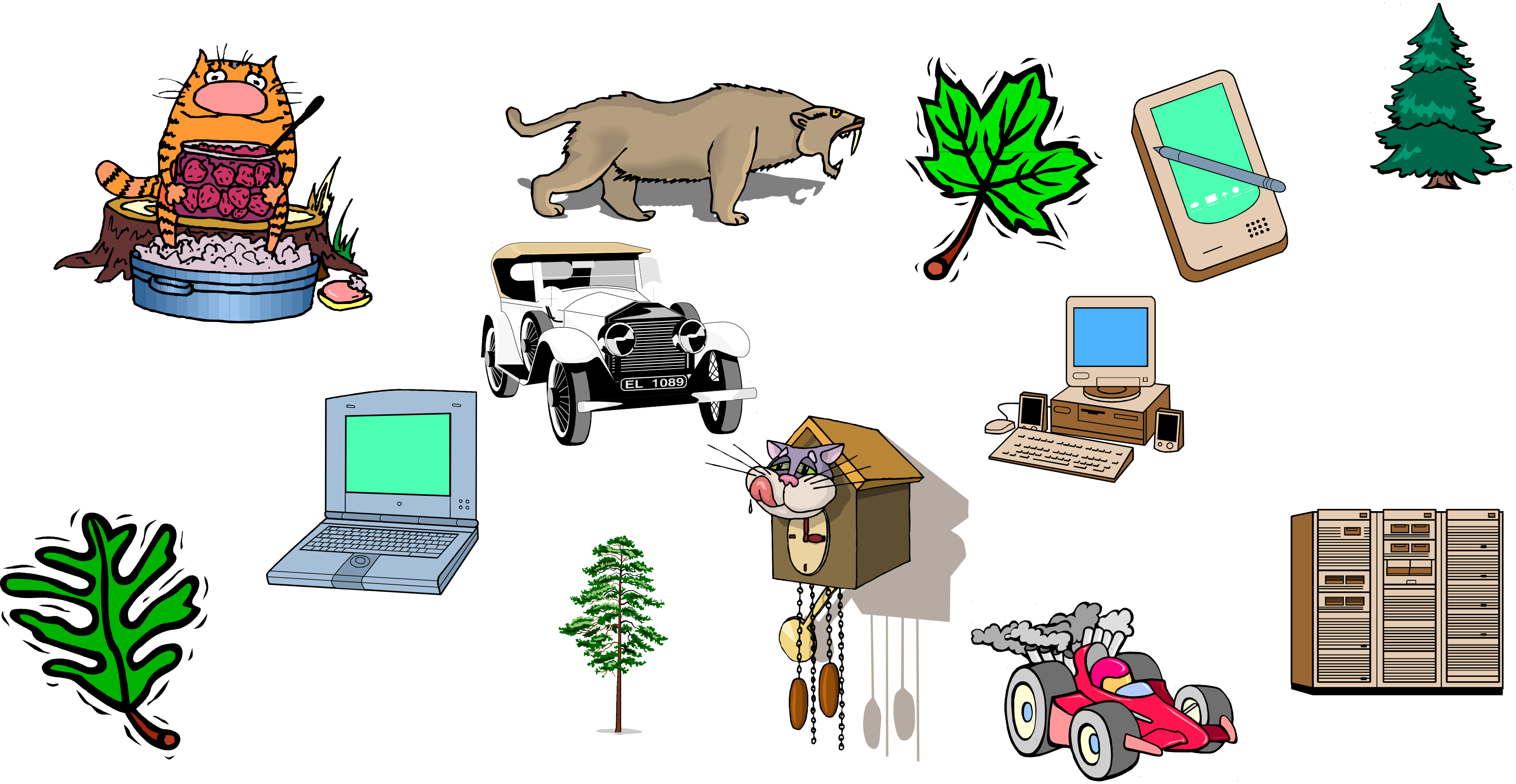
- All objects of a particular class have the same set of operations. They are not shown on the object diagram, they are shown on the class diagram (see later)
- Attribute types are often omitted to simplify the diagram
- Naming:
  - object and attribute names in lowerCamelCase
  - class names in UpperCamelCase

What are classes?

- Every object is an instance of one class - the class describes the "type" of the object
- Classes allow us to model sets of objects that have the *same* set of features - a class acts as a template for objects:
  - The class determines the structure (set of features) of all objects of that class
  - All objects of a class *must* have the same set of operations, *must* have the same attributes, but *may* have different attribute values
- Classification is one of the most important ways we have of organising our view of the world
- Think of classes as being like:
  - Rubber stamps
  - Cookie cutters

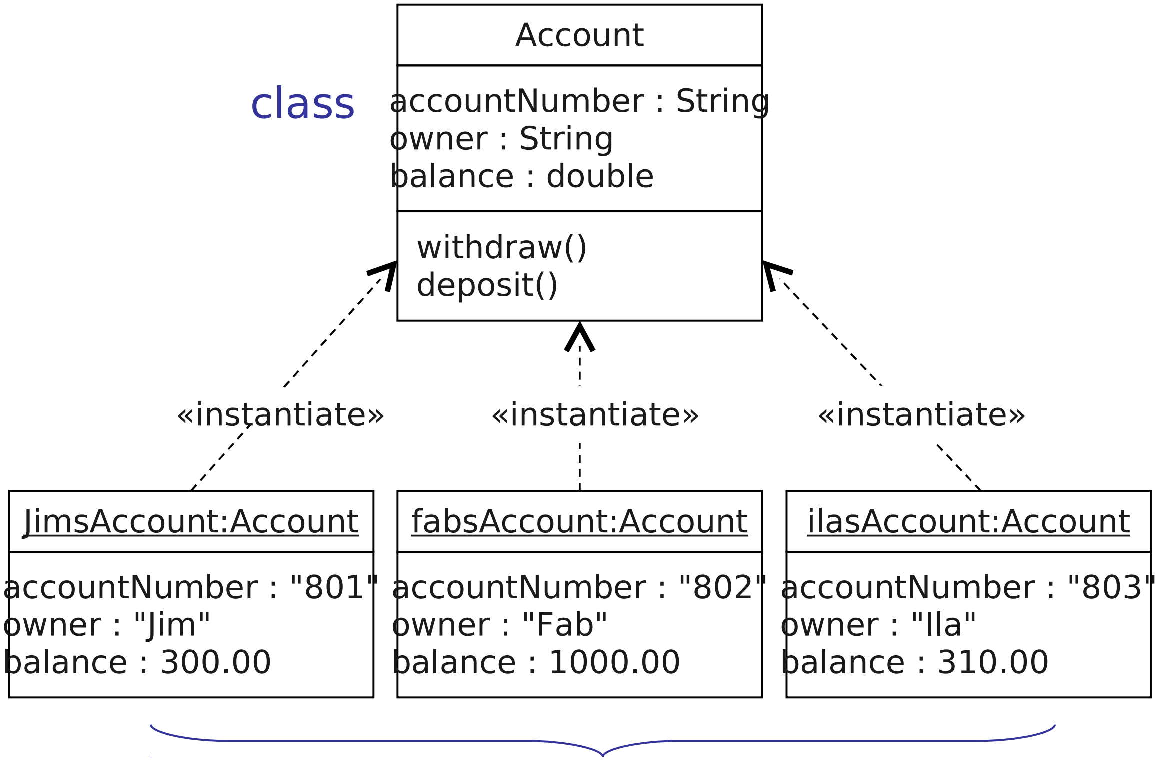


Exercise - how many classes?



Classes and objects

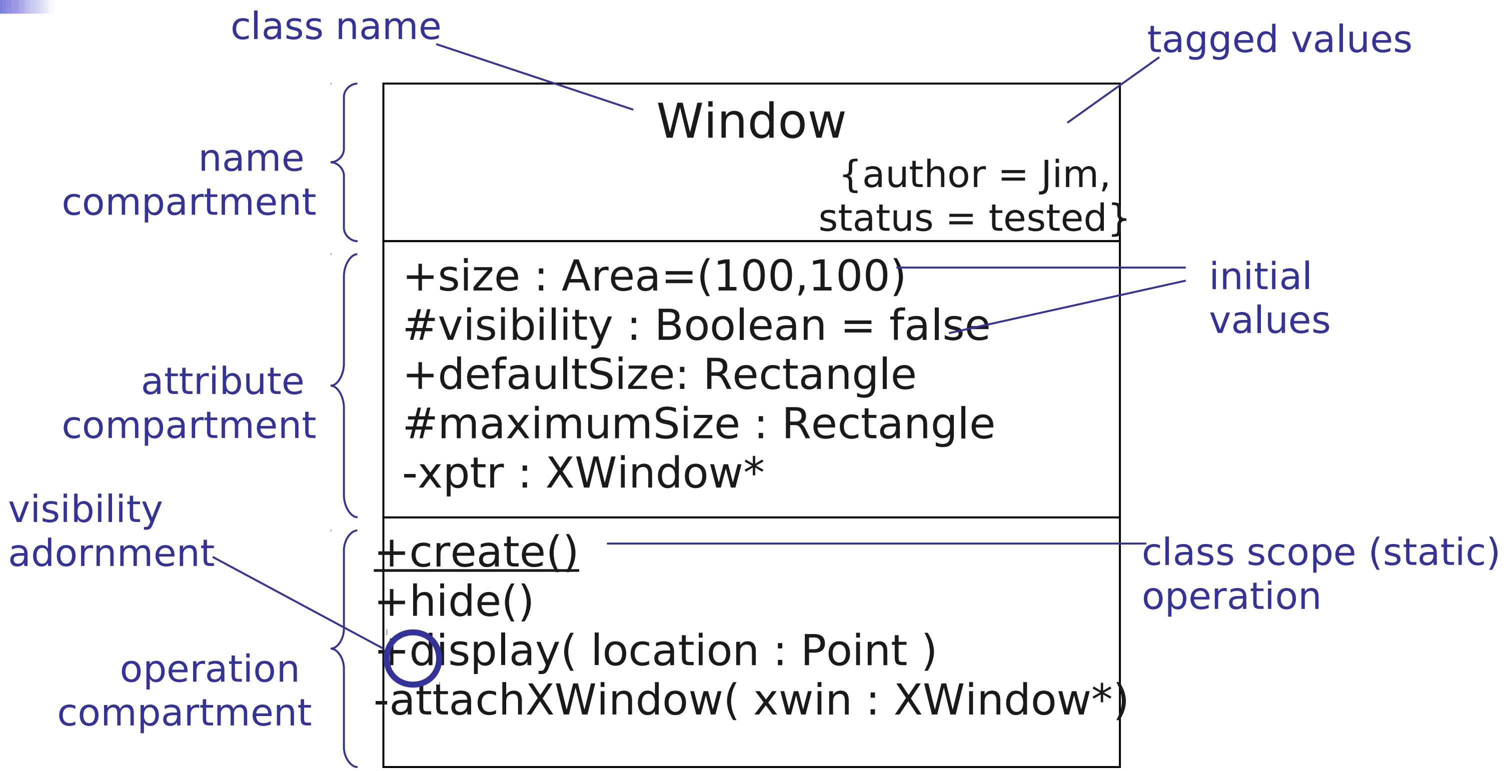
- Objects are instances of classes
- Instantiation is the creation of new instances of model elements
- Most classes provide special operations called *constructors* to create instances of that class. These operations have class-scope i.e. they belong to the class itself rather than to objects of the class
- We will see instantiation used with other modelling elements later on



objects

objects are instances of classes

UML class notation



class name

tagged values

name compartment

Window
{author = Jim,
status = tested}

attribute compartment

+size : Area=(100,100)
#visibility : Boolean = false
+defaultSize: Rectangle
#maximumSize : Rectangle
-xptr : XWindow*

initial values

visibility adornment

operation compartment

+create()
+hide()
+display(location : Point)
-attachXWindow(xwin : XWindow*)

class scope (static) operation

- Classes are named in UpperCamelCase
- Use descriptive names that are nouns or noun phrases
- Avoid abbreviations!

7.5.2

Attribute compartment

visibility name : type multiplicity = initialValue

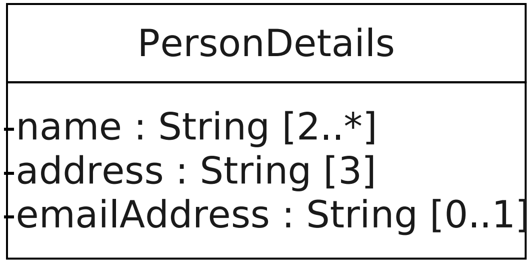
mandatory

- Everything is optional except name
- initialValue is the value the attribute gets when objects of the class are instantiated
- Attributes are named in lowerCamelCase
  - Use descriptive names that are nouns or noun phrases
  - Avoid abbreviations
- Attributes may be prefixed with a stereotype and postfixed with a list of tagged values

7.5.2.1

Visibility

Symbol	Name	Semantics
+	public	Any element that can access the class can access any of its features with public visibility
-	private	Only operations within the class can access features with private visibility
#	protected	Only operations within the class, or within children of the class, can access features with protected visibility
~	package	Any element that is in the same package as the class, or in a nested subpackage, can access any of its features with package visibility



PersonDetails

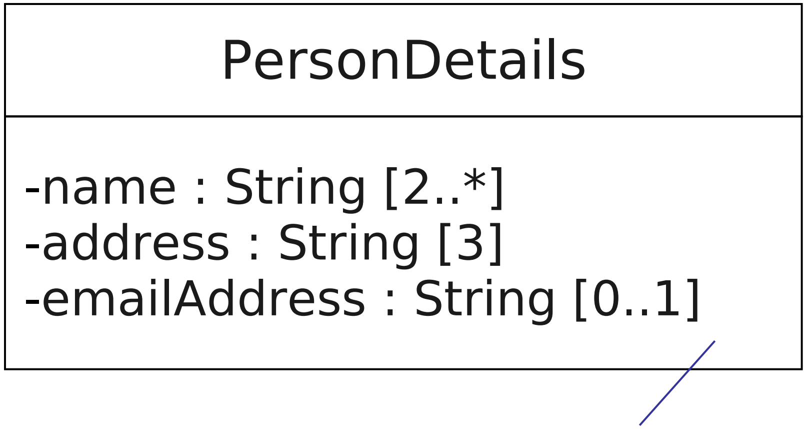
-name : String [2..*]
-address : String [3]
-emailAddress : String [0..1]

- You may ignore visibility in analysis
- In design, attributes usually have private visibility (encapsulation)

7.5.2.3

Multiplicity

- Multiplicity allows you to model collections of things
  - [0..1] means an that the attribute may have the value null



PersonDetails

-name : String [2..*]
-address : String [3]
-emailAddress : String [0..1]

name is composed of 2 or more Strings

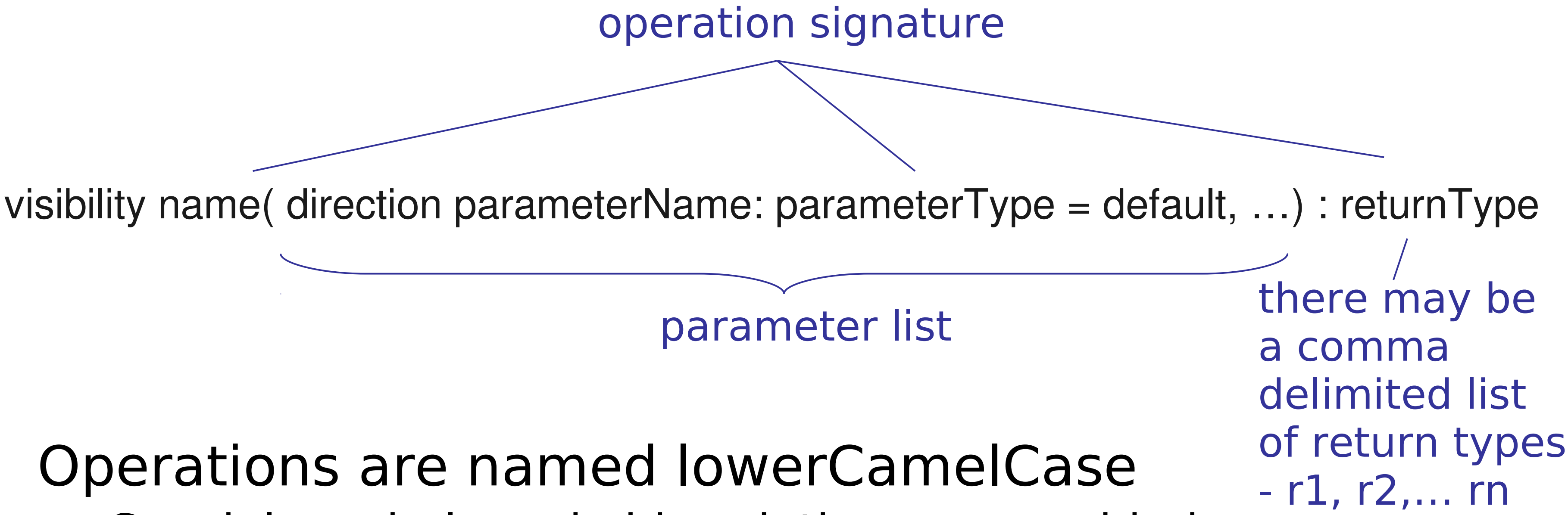
address is composed of 3 Strings

emailAddress is composed of 1 String or null

multiplicity expression

7.5.3

Operation compartment



operation signature

visibility name(direction parameterName: parameterType = default, …) : returnType

parameter list

there may be a comma delimited list of return types - r1, r2,… rn

- Operations are named lowerCamelCase
  - Special symbols and abbreviations are avoided
  - Operation names are usually a verb or verb phrase
- Operations may have more than one returnType
  - They can return multiple objects (see next slide)
- Operations may be prefixed with a stereotype and postfixed with a list of tagged values

7.5.3.1

Parameter direction

use in detailed design only!

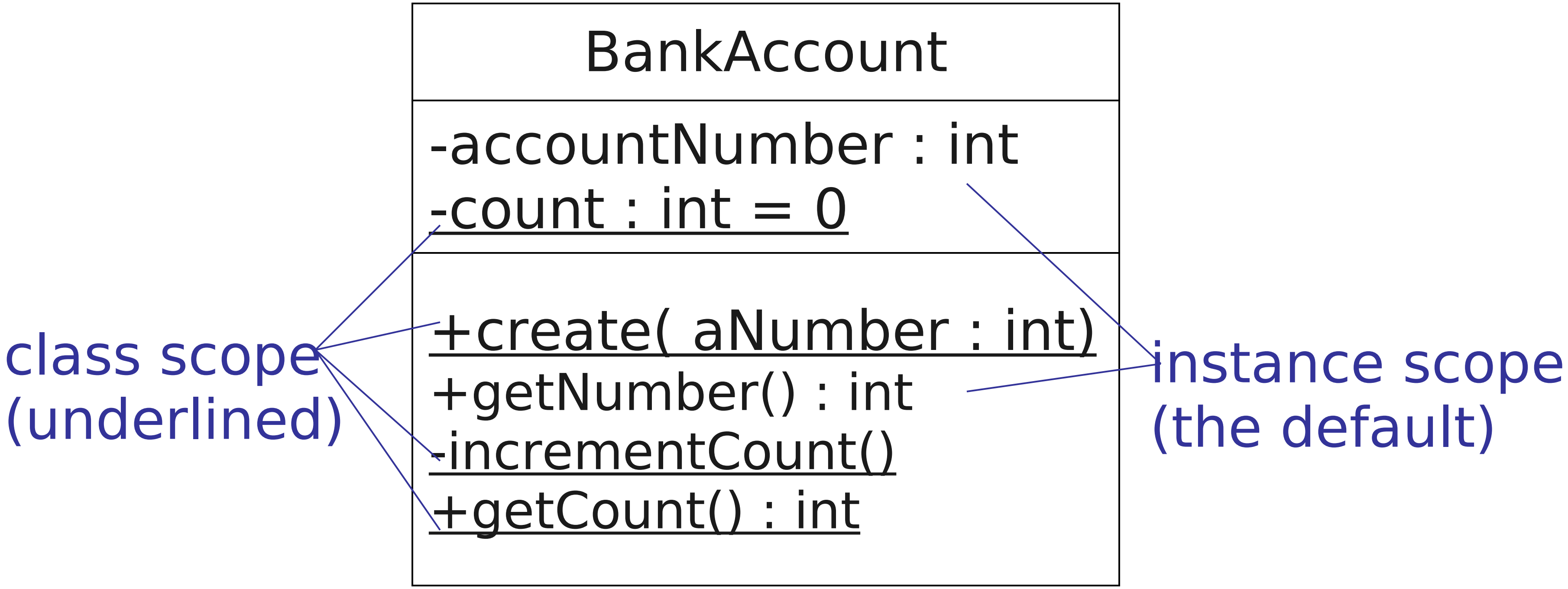
parameter direction	semantics
in	the parameter is an input to the operation. It is not changed by the operation. This is the default
out	the parameter serves as a repository for output from the operation
inout	the parameter is an input to the operation and it may be changed by the operation
return	the parameter is one of the return values of the operation. An alternative way of specifying return values

example of multiple return values:

```
maxMin( in a: int, in b:int, return maxValue:int return minValue:int )
…
max, min = maxMin( 5, 10 )
```

Scope

- There are two kinds of scope for attributes and operations:



BankAccount

-accountNumber : int
-count : int = 0

+create(aNumber : int)
+getNumber() : int
-incrementCount()
+getCount() : int

class scope (underlined)

instance scope (the default)

7.6.1

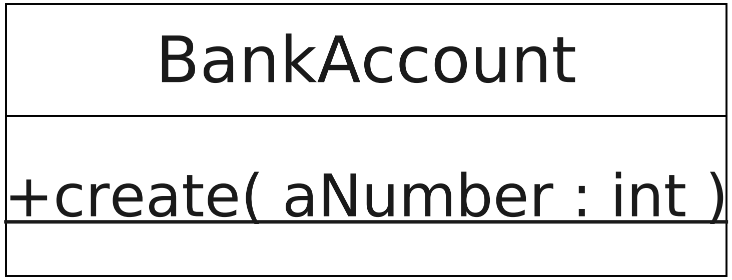
Instance scope vs. class scope

	instance scope	class scope
attributes	By default, attributes have instance scope	Attributes may be defined as class scope
	Every object of the class gets its own copy of the instance scope attributes	Every object of the class shares the same, single copy of the class scope attributes
	Each object may therefore have different instance scope attribute values	Each object will therefore have the same class scope attribute values
operations	By default, operations have instance scope	Operations may be defined as class scope
	Every invocation of an instance scope operation applies to a specific instance of the class	Invocation of a class scope operation does not apply to any specific instance of the class – instead, you can think of class scope operations as applying to the class itself
	You can't invoke an instance scope operation unless you have an instance of the class available. You can't use an instance scope operation of a class to create objects of that class, as you could never create the first object	You can invoke a class scope operation even if there is no instance of the class available – this is ideal for object creation operations

scope determines access

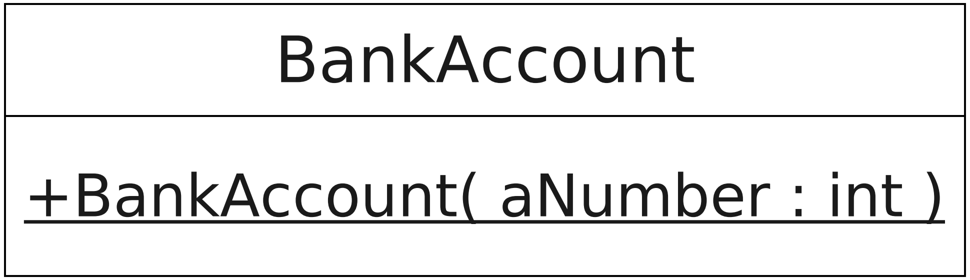
Object construction

- How do we create instances of classes?
- Each class defines one or more class scope operations which are *constructors*. These operations create new instances of the class


BankAccount

+create(aNumber : int)

generic constructor name

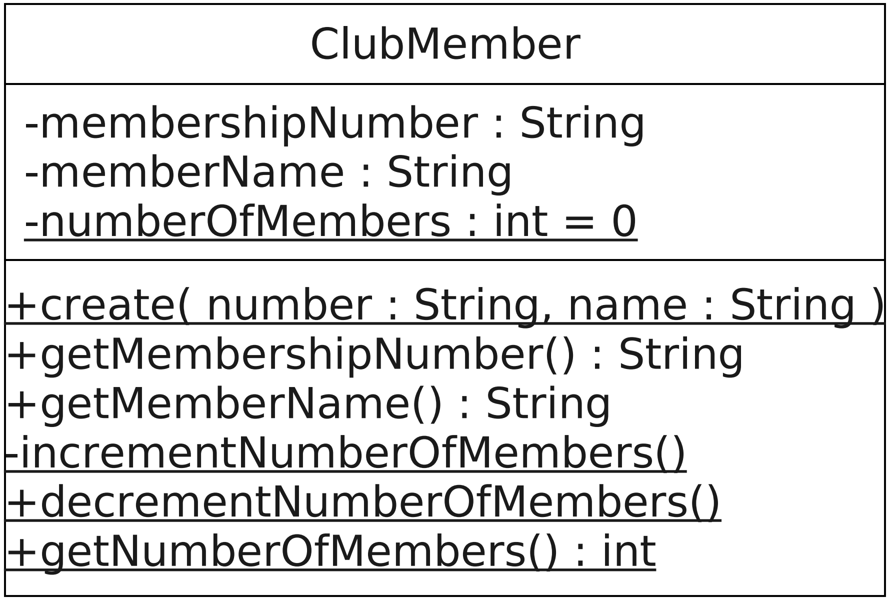

BankAccount

+BankAccount(aNumber : int)

Java/C++ standard

ClubMember class example

- Each ClubMember object has its own copy of the attribute membershipNumber
- The numberOfMembers attribute exists only once and is shared by all instances of the ClubMember class
- Suppose that in the create operation we increment numberOfMembers:
  - What is the value of count when we have created 3 account objects?



ClubMember

-membershipNumber : String
-memberName : String
-numberOfMembers : int = 0

+create(number : String, name : String)
+getMembershipNumber() : String
+getMemberName() : String
-incrementNumberOfMembers()
+decrementNumberOfMembers()
+getNumberOfMembers() : int

Summary

- We have looked at objects and classes and examined the relationship between them
- We have explored the UML syntax for modelling classes including:
  - Attributes
  - Operations
- We have seen that scope controls access
  - Attributes and operations are normally instance scope
  - We can use class scope operations for constructor and destructors
  - Class scope attributes are shared by all objects of the class and are useful as counters