Software Engineering Department
School of Information & Communication Technology
Hanoi University of Science and Technology

Object-Oriented Programming

Encapsulation and Writing Class, Creating and Using Objects

Cao Tuấn Dũng

Goals

- Principle and role of abstraction
- Introduction to encapsulation and information hiding
- Class construction
  - Class definition
  - Creating methods, fields/properties
- Creating and using objects
  - Construction
  - Object declaration and initialization
  - Object usage

Outline

1. Data abstraction
2. Encapsulation and Class construction
3. Object creation and usage

1.1 Review on Abstraction

- *Any model that includes the most important, essential, or distinguishing aspects of something while suppressing or ignoring less important, immaterial, or diversionary details. The result of removing distinctions so as to emphasize commonalties* (Dictionary of Object Technology, Firesmith, Eykholt, 1995).
- Reduce and factor out details so that one can focus on a few concepts at a time
  - *"abstraction – a concept or idea not associated with any specific instance".*
  - Example: Mathematics definition

Why abstraction?

- Programs are complex.
  - Windows XP: ~45 million lines of code
  - Mathematica: over 1.5 million

- Abstraction helps
  - *Many-to-one* – "forget the details"
    - Allow managing a complex problem by focusing on important properties of an entity in order to distinguish with other entities
  - Separate "what" from "how"

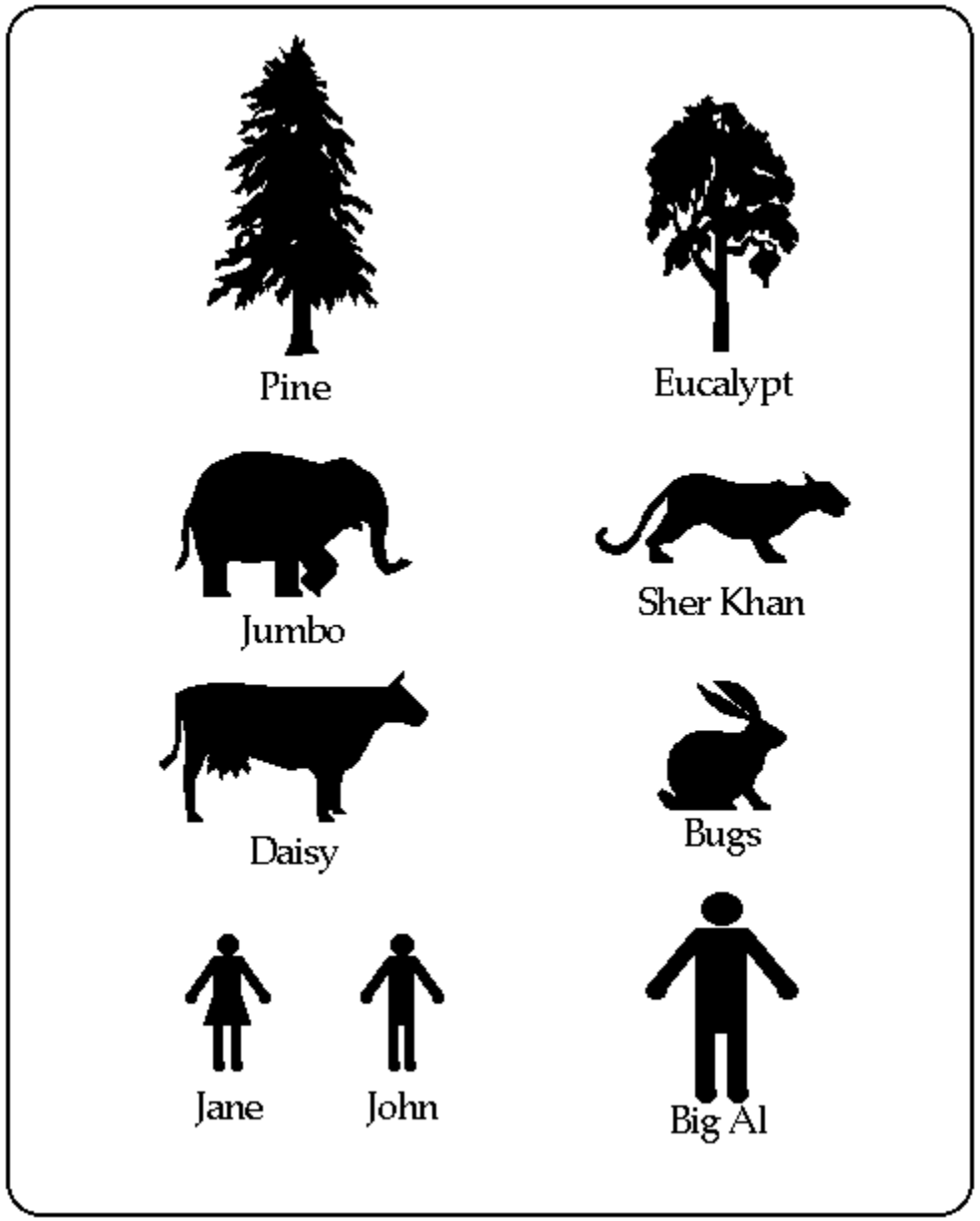
Abstraction (2)

- Abstraction is the process of extracting common features from specific examples

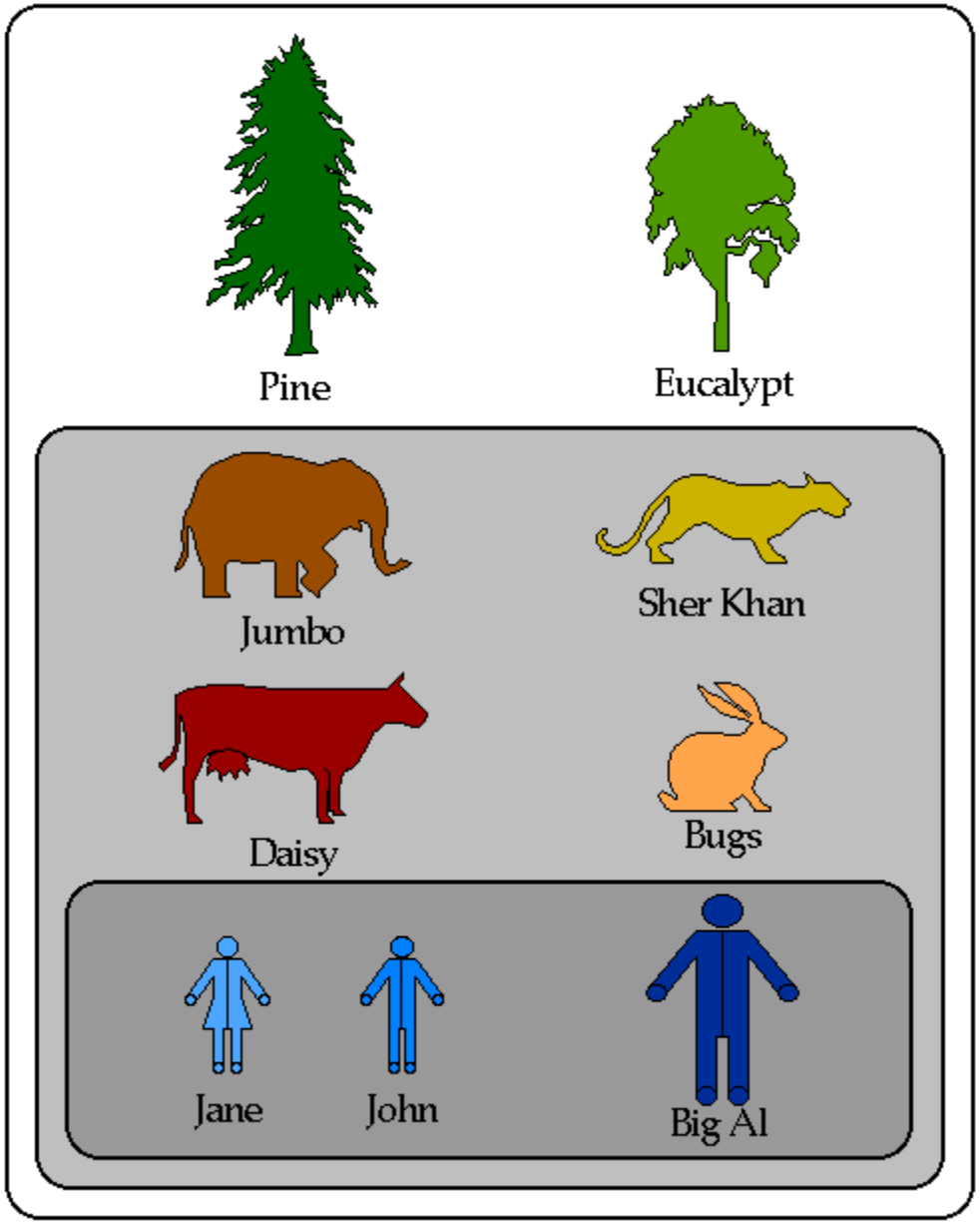
- Abstraction is a view or representation of an entity containing only properties related to some context

-

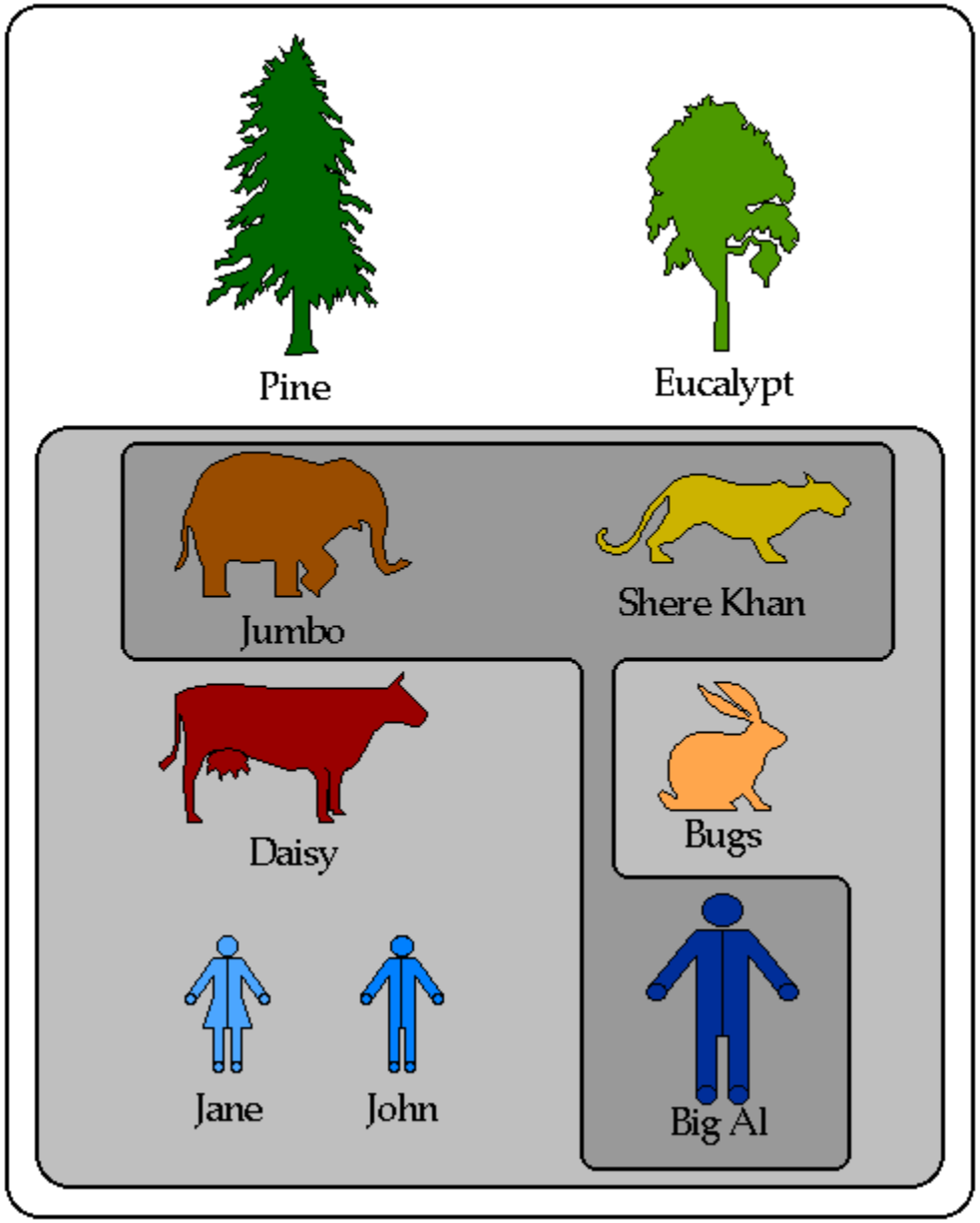
unclassified "things"



- organisms, mammals, humans



- organisms, mammals, dangerous mammals



1.2. Types of abstraction

- **Control abstraction** implies a program control mechanism without specifying internal details.
  - subprogram and control flow, exception handling
  - Example: a := (1 + 2) * 5
    - Without control abstraction, programmers have to specify all the registers, binary-level steps…
- **Functional (Procedural) abstraction** hide details associated with executing an operation or task.
  - Declare a function by giving a name to a piece of code.
  - Other programmers can use your abstraction by invoking the function.

1.2 Types of abstraction (cont..)

- **Data abstraction**:
  - One of the most powerful programming paradigms
  - Used as a tool to increase the modularity of a program

- An abstraction that **hides how a data object is represented**
- Data abstraction
  - Define data objects for problem.
  - Hide representation.
  - *What* you can do with the data is separated from *how* it is represented

1.3 Data Abstraction

- It is used to build walls between a program and its data structures
- Focus on the meaning of the operations (behavior), to avoid over-specification.
- The representation details are confined to only a small set of procedures that create and manipulate data, and all other access is indirectly via only these procedures.

Data Abstraction : Motivation

- Client/user perspective
  - Interested in ***what*** a program does, not *how*.
  - Minimize irrelevant details for clarity.
  - *Representation Independence*.
- Server/implementer perspective
  - Restrict users from making *unwarranted assumptions* about the implementation.
  - Reserve right to change representation to improve performance, … (*maintaining behavior*).
  - *Information Hiding*.

1.4 Abstract Data Types

- Abstract Data Type (ADT): data type that specifies logical properties without the implementation details
- Queues (empty, enQueue, deQueue, isEmpty)
  - array-based implementation
  - linked-list based implementation
- Tables (empty, insert, lookUp, delete, isEmpty)
  - Sorted array (logarithmic search)
  - Hash-tables (*ideal*: constant time search)
  - AVL trees (height-balanced)
  - B-Trees (optimized for secondary storage)

ADT

- An abstract data type is any type you want to add to the language over and above the fundamental types
- For example, you might want to add a new type called: list
  - which maintains a list of data
  - the data structure might be an array of structures
  - operations might be to add to, remove, display all, display some items in the list

ADT

- An ADT has three properties
  - Name of the ADT (type name)
  - Set of values belonging to the ADT (domain)
  - Set of operations on the data

Example of Abstract Data Type

- Type: complex numbers
- Domain: numbers of the form `a + bi`
- Operations
  - add, subtract
  - multiply
  - divide

$(4 + 2i) + (6 - 3i)$

$(7 - 2i) * (5 + 4)$

$$\frac{(3 - 2i)}{5i}$$

Data Abstraction

- Think of a class as similar to a data type
  - and an object as a variable
- And, just as we can have zero or more variables of any data type…
  - we can have zero or more objects of a class!
- Then, we can perform operations on an object in the same way that we can access members of a struct…

1.6 Data abstraction in OOP

- Objects in reality are very complex

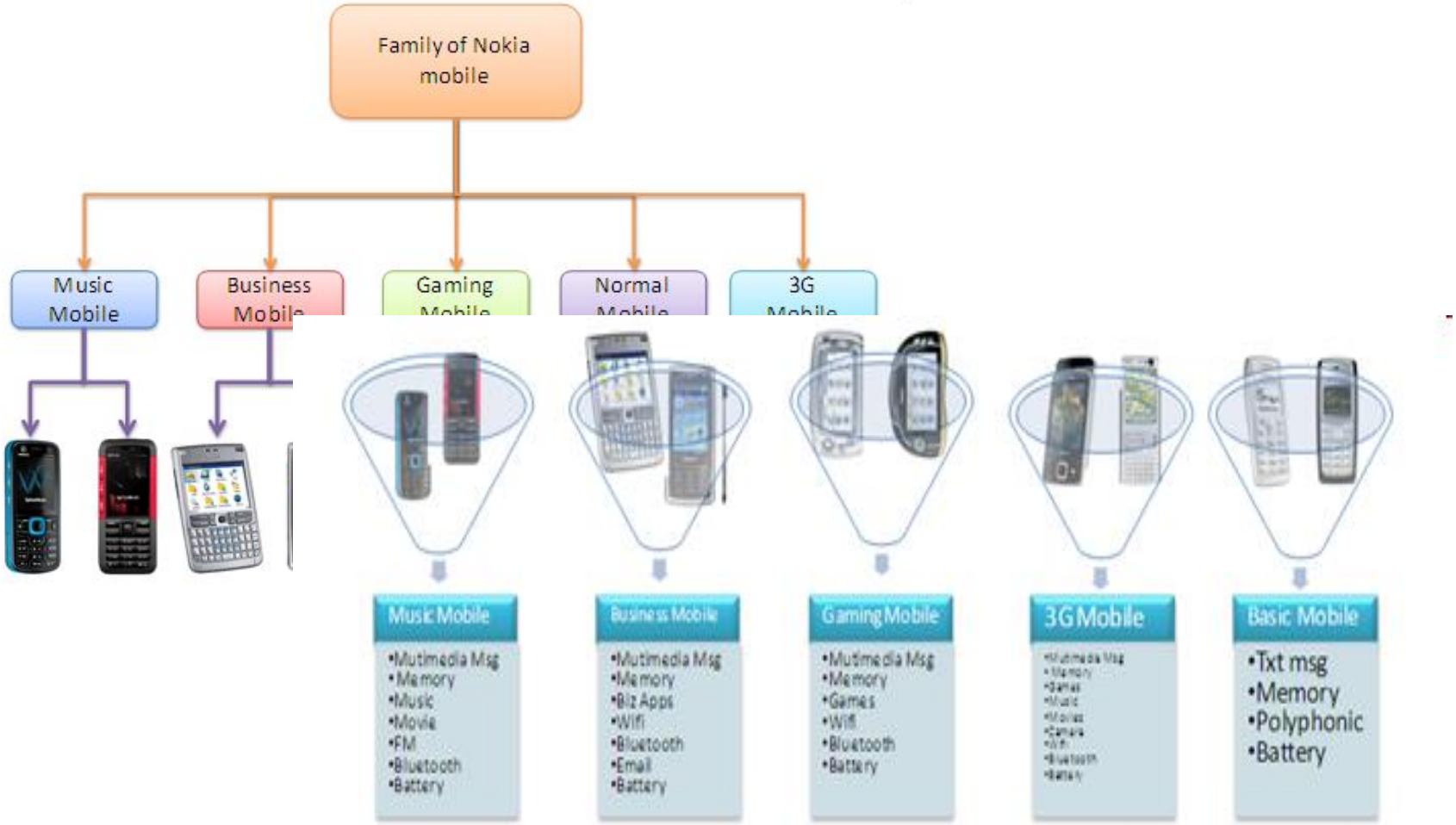
- Need to be simplified by ignoring all the unnecessary details
- Only “extract” related/involving, important information to the problem

Example: Abstract the following

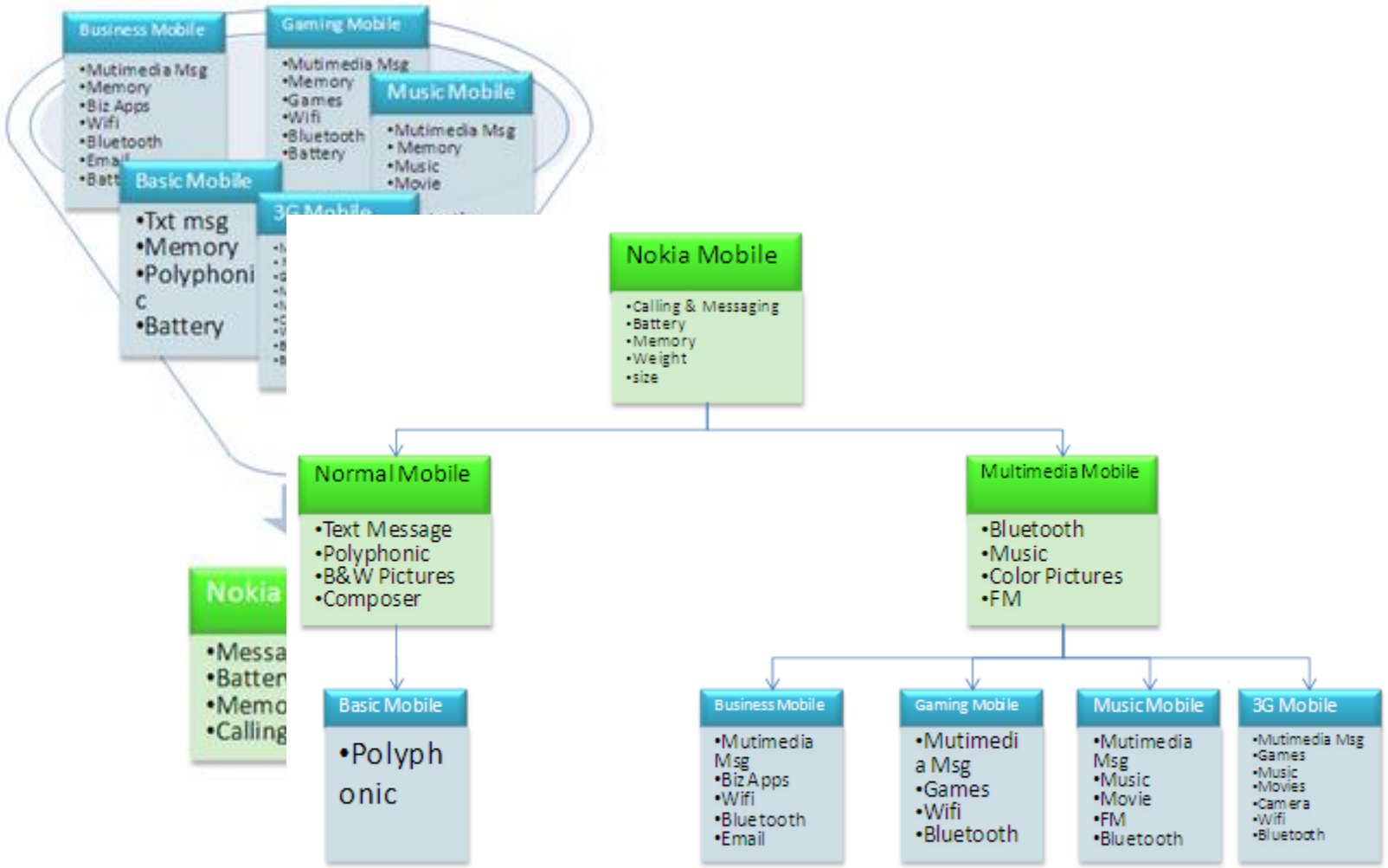
- What are the common properties of these entities? What are particular properties?
  - All are Nokia phone
  - Sliding, folding, …
  - Phone for Businessman, Music, 3G
  - QWERTY keyboard, Basic Type, No-keyboard type
  - Color, Size, …

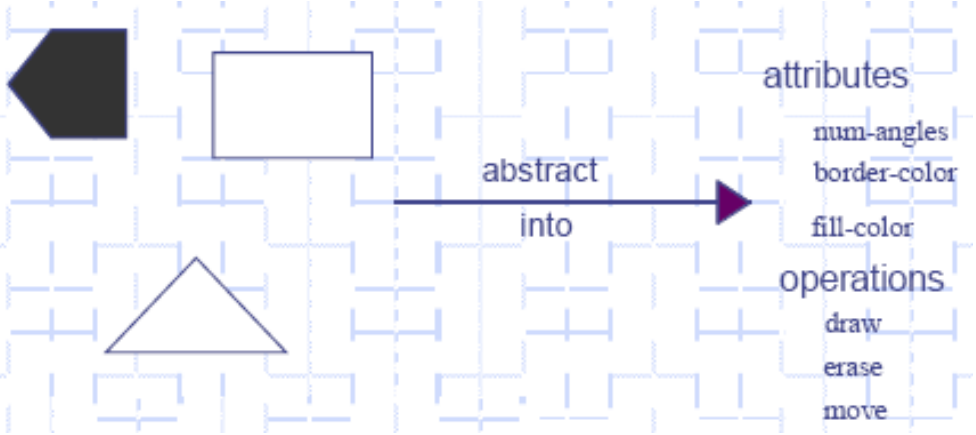

Family of Nokia mobile

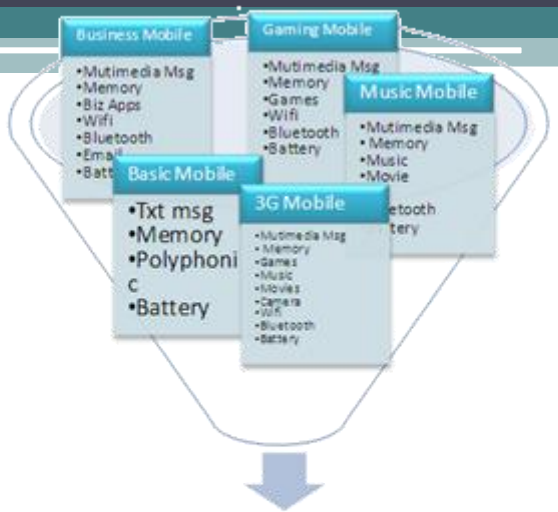
Categorizing mobile – Specifying non-common characteristics



From details to generalization




attributes
num-angles
border-color
fill-color
abstract
into
operations
draw
erase
move


Business Mobile
•Mutimedia Msg
•Memory
•Biz Apps
•Wifi
•Bluetooth
Gaming Mobile
•Mutimedia Msg
•Memory
•Games
•Wifi
•Bluetooth
•Battery
Music Mobile
•Mutimedia Msg
• Memory
•Music
•Movie
Basic Mobile
•Txt msg
•Memory
•Polyphonic
•Battery
3G Mobile


Nokia Mobile
•Messaging
•Battery
•Memory
•Calling


Family of Nokia mobile

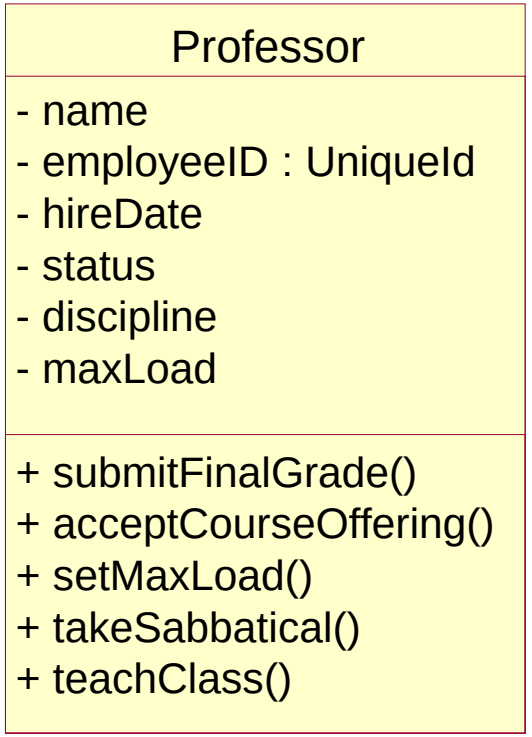
1.7 Class vs. Objects

- Class is concept model, describing entities
- Class is a prototype, defining common properties and methods of objects
- A class is an abstraction of a set of objects.

- Objects are real entities
- Object is a representation (instance) of a class, *data of different objects are different*
- Each object has a class specifying its data and behavior.

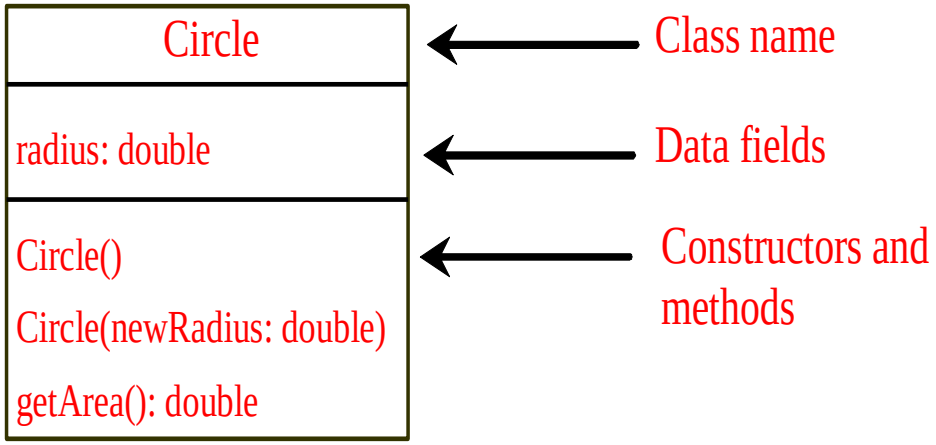
Class representation in UML

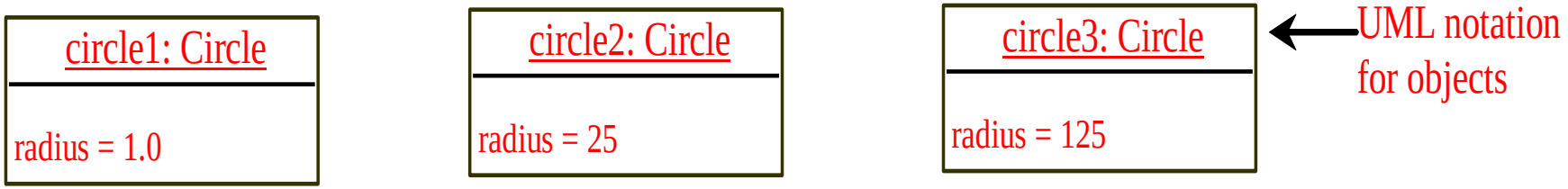
- Class is represented by a rectangle with three parts:
  - Class name
  - Structure (Properties)
  - Behavior (operation)



UML Class Diagram

UML Class Diagram

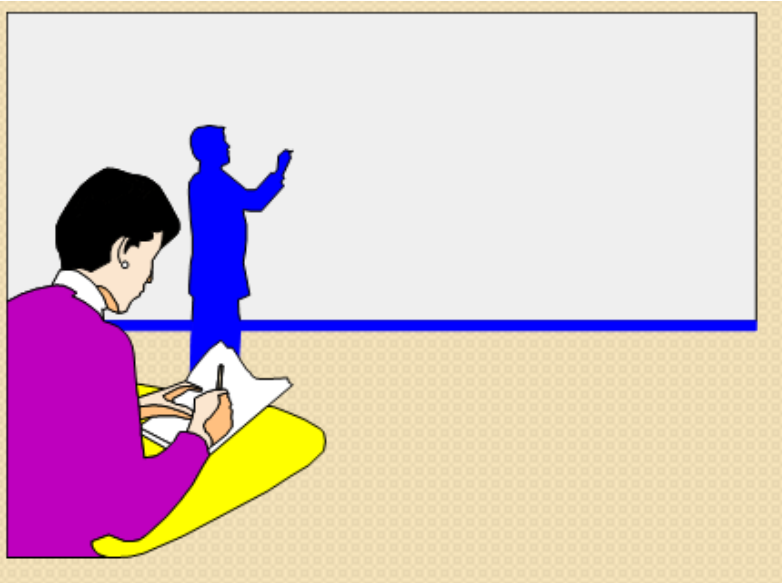




A Sample Class

Class Course

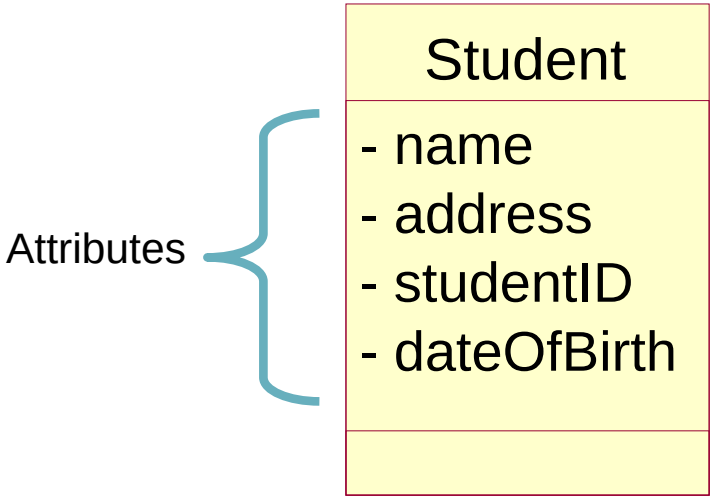
Properties
Name
Location
Days offered
Credit hours
Start time
End time



Behavior
Add a student
Delete a student
Get course roster
Determine if it is full

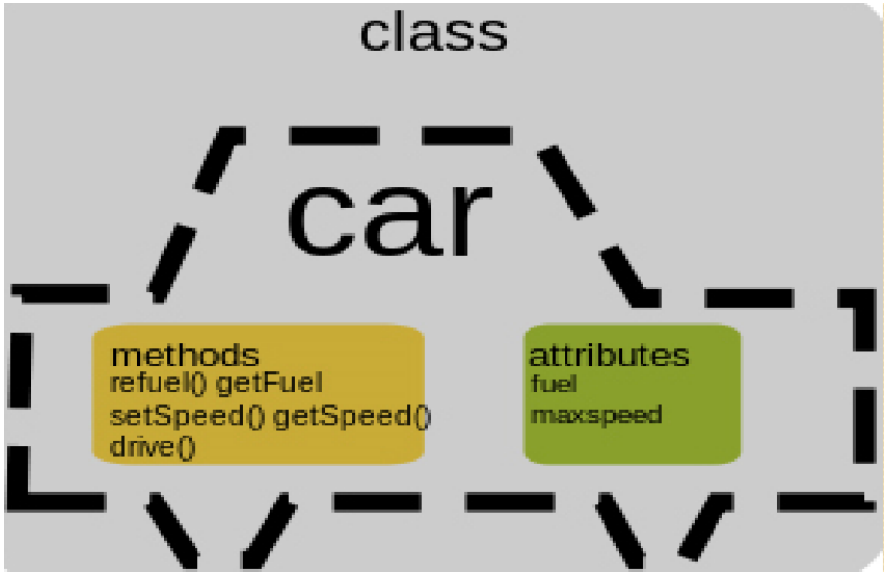
What is attribute?

- Attributes - things that the object stores data in, generally variables.
- An attribute is a named characteristic of a class specifying a value range of its representations.
  - A class might have no property or any number of properties.

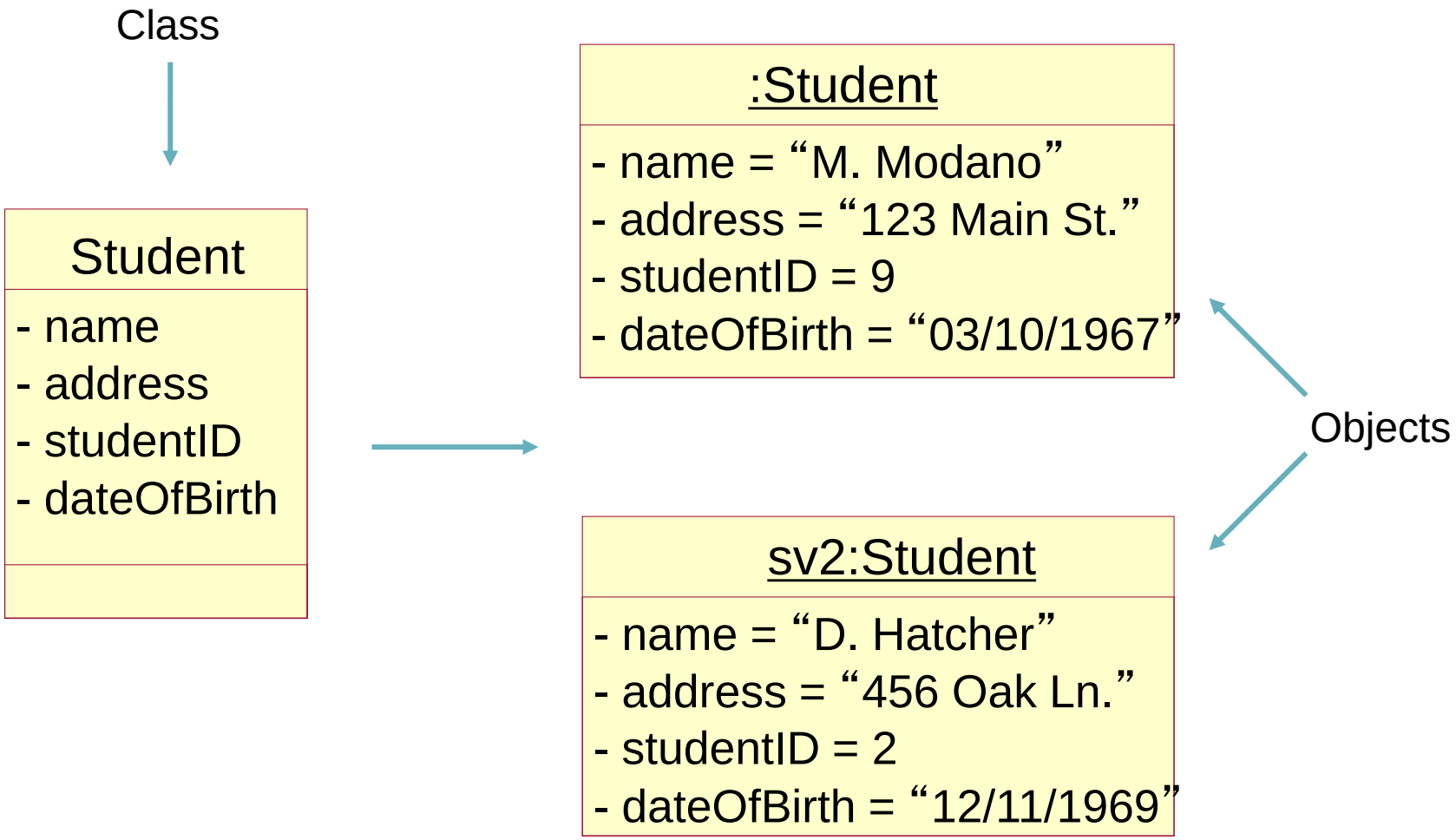


What Is an Operation(Method)?

- An operation is the implementation of a service that can be requested from any object of the class to affect behavior.
- Methods - Functions and Procedures attached to an Object and allowing the object to perform actions.
- A class may have any number of operations or none at all.



Class and Object in UML



Object Recall

- Object is a representation of en entity, or in reality such as a table, a chair, a human, or an abstract entity.
- ***Object is an abstraction that has clear and important definition to the application.***

- Every object in the system always has three descriptions::
  - State
  - Behavior
  - ***Identity charateristic***

State of Object

- An object state is one the the contexts in which the object exists. Normally, object state is changed as time goes by

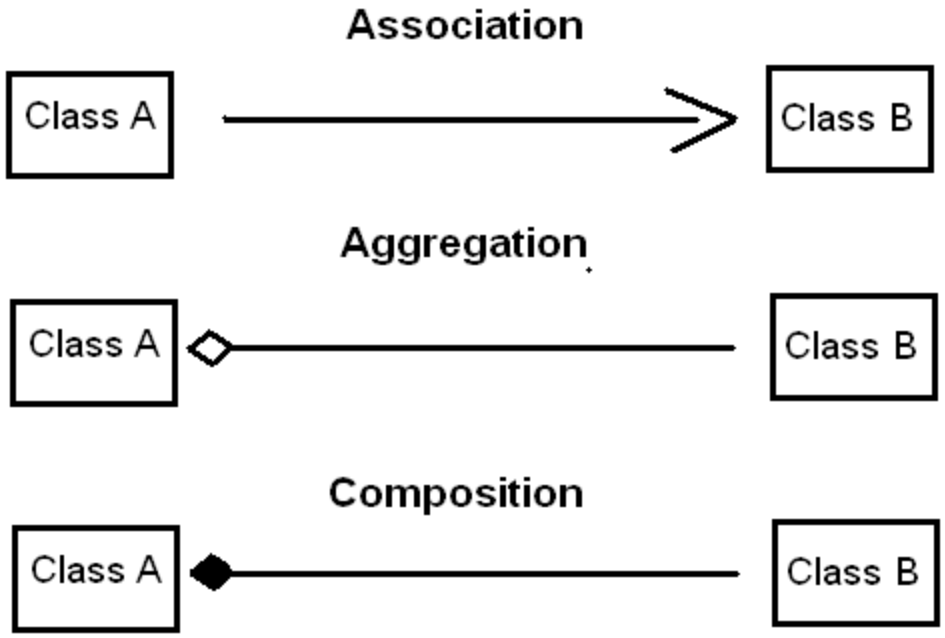
- A stat object is defined as a set of all the attributes, values of these attributes, and the relations between the object with other objects.

Behavior of Object

- Object behavior defines how it responses to requests from other objects and that's all an object can do. Object behavior is performed by a set of operations.
- **Identity characteristic**
  - Identity characteristic is an attribute of object that allows distinguishing with other objects.

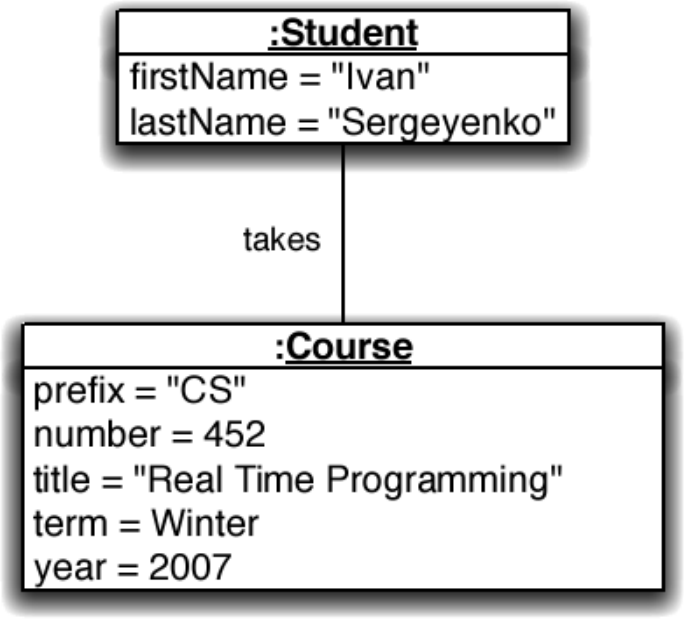
1.8 Relations between objects

- A system is built from many classes and objects.
- There are two kinds of relation between objects:
  - *link*
  - *aggregation*



Link relation

- **Link relation** is a physic or logic link between objects. An object coordinates with other objects via its links to these objects. A link represents a specific association, in which an object (client) is using services provided by other objects (supplier).

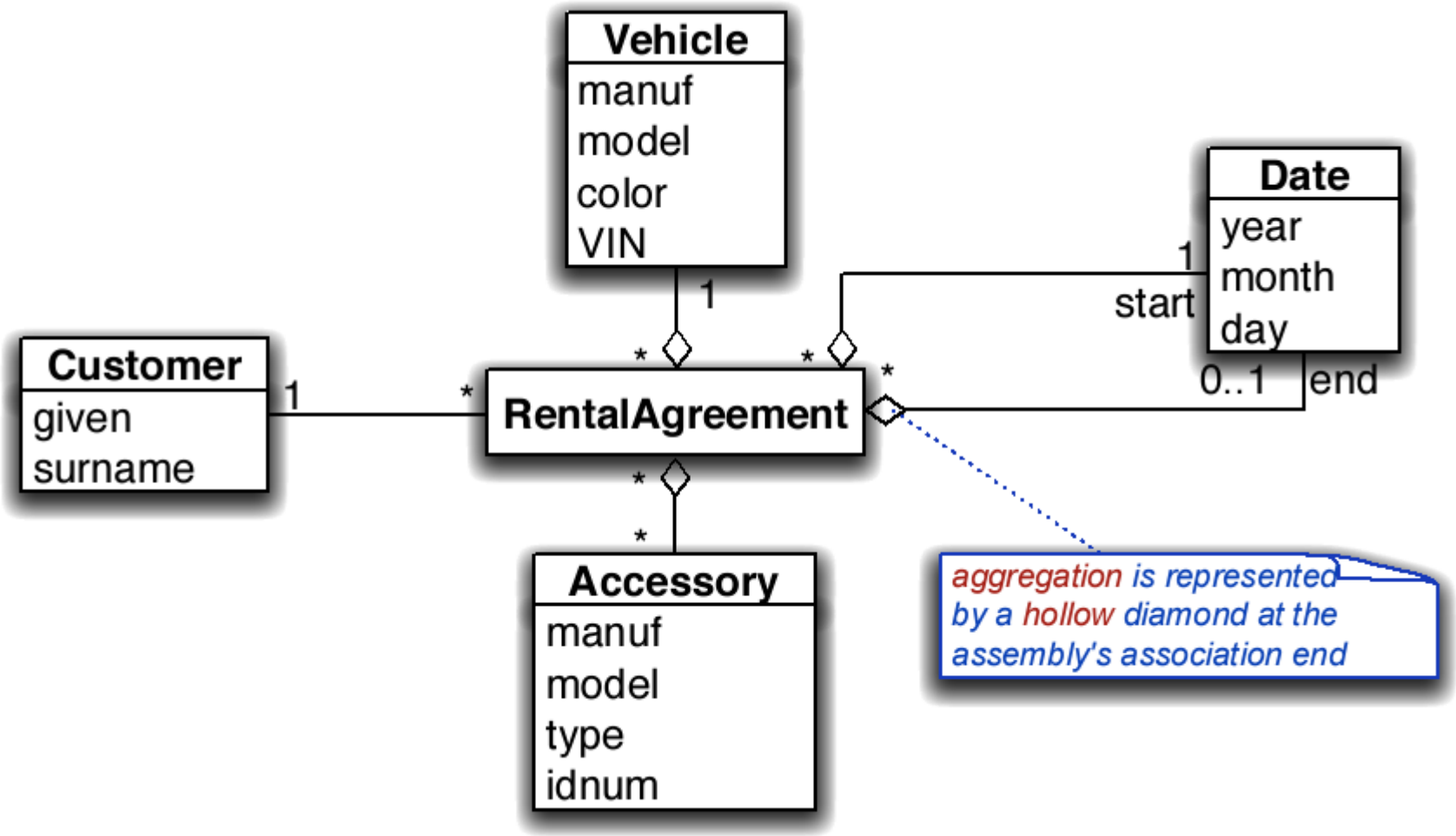


Link relation

- *Actor*: An object can operate on other objects and are not manipulated by other objects.
- *Server*: An object that is never able to operate on other objects; it can only be manipulated by other objects.
- *Agent*: An object that can not only operates on other objects but also be manipulated by other objects.

Aggregation relation

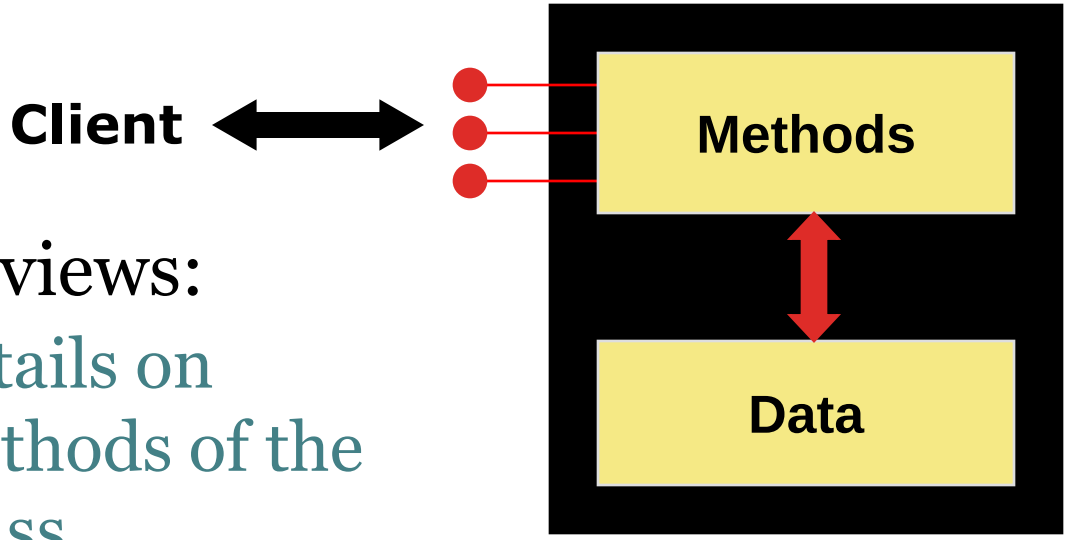
- An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object.
- For example, a car has 4 wheels, a steering, an engine,...


Vehicle
manuf
model
color
VIN
Date
year
month
day
1
start
0..1
end
Customer
given
surname
1
*
RentalAgreement
Accessory
manuf
model
type
idnum
aggregation is represented by a hollow diamond at the assembly's association end

Outline

1. Data abstraction
2. Encapsulation and Class construction
3. Object creation and usage

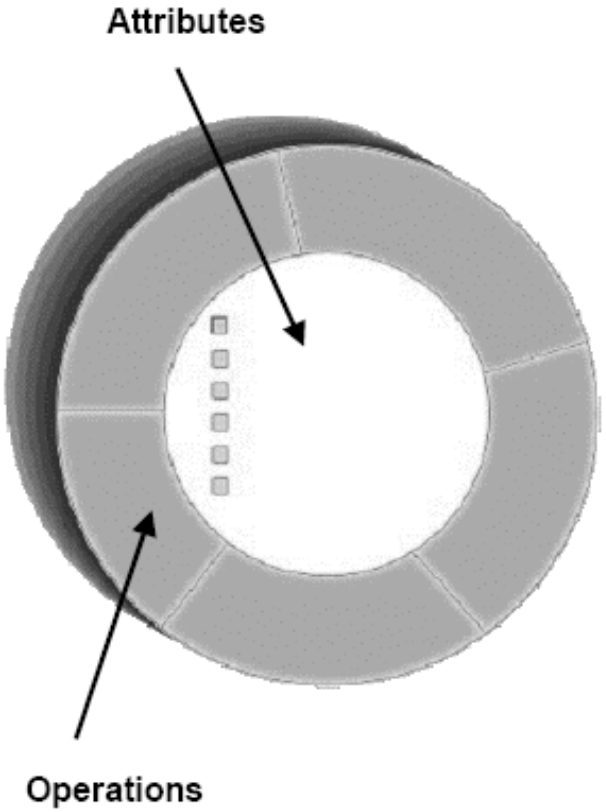
2.1. Encapsulation



- An object has two views:
  - Internal view: Details on attributes and methods of the corresponding class
  - External view: Services provided by the object and how the object communicates with all the rest of the system

2.1. Encapsulation (2)

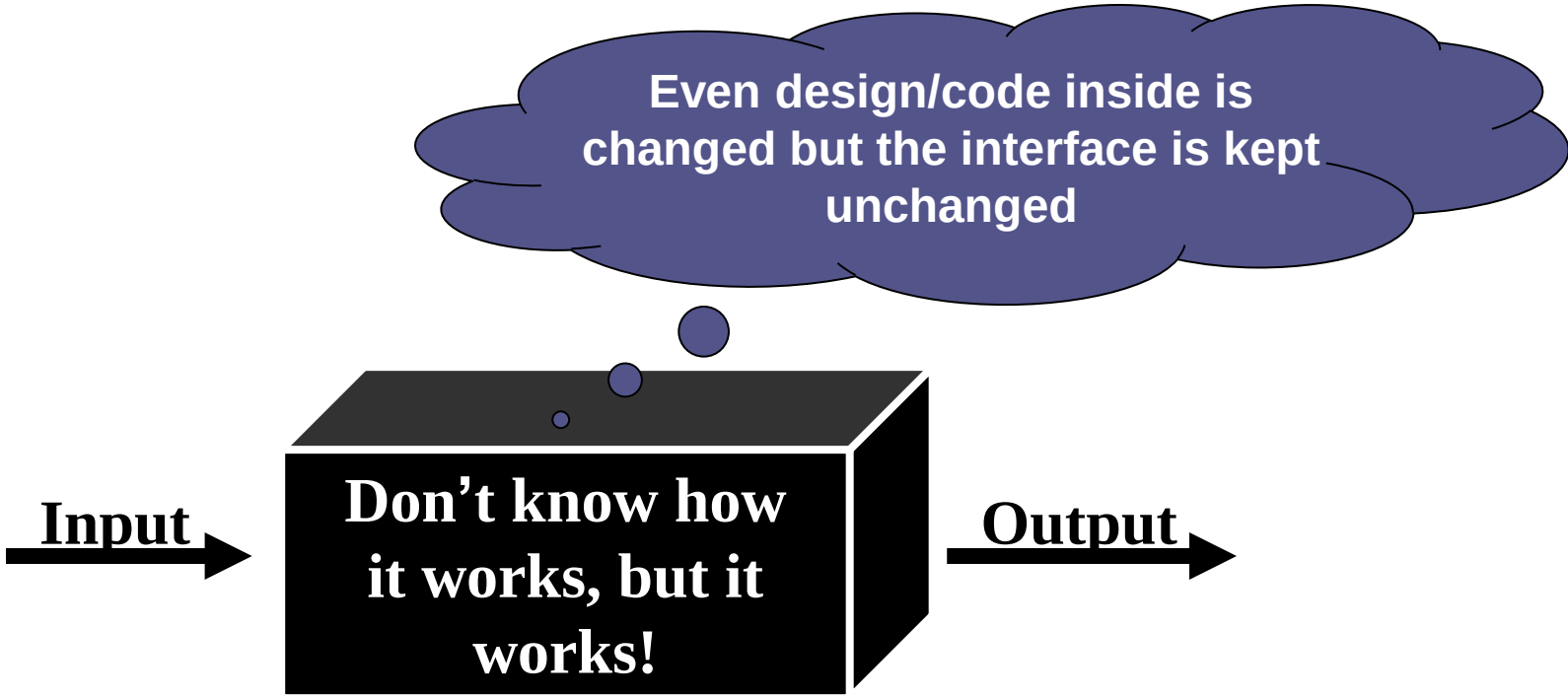
- Data/properties and behavior/methods are encapsulated in a class → Encapsulation



BankAccount
- owner: String - balance: double
+ debit(double): boolean +credit(double)

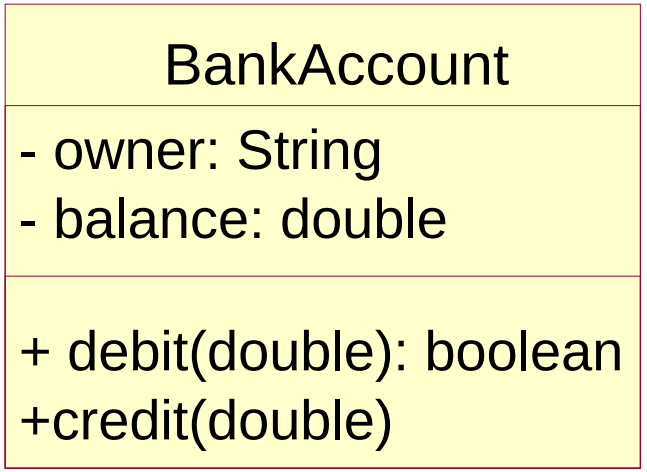
2.1. Encapsulation (3)

- An object is an encapsulated entity that supplies a set of services
- An encapsulated object can be considered as a black box – all the internal operations are hidden to client



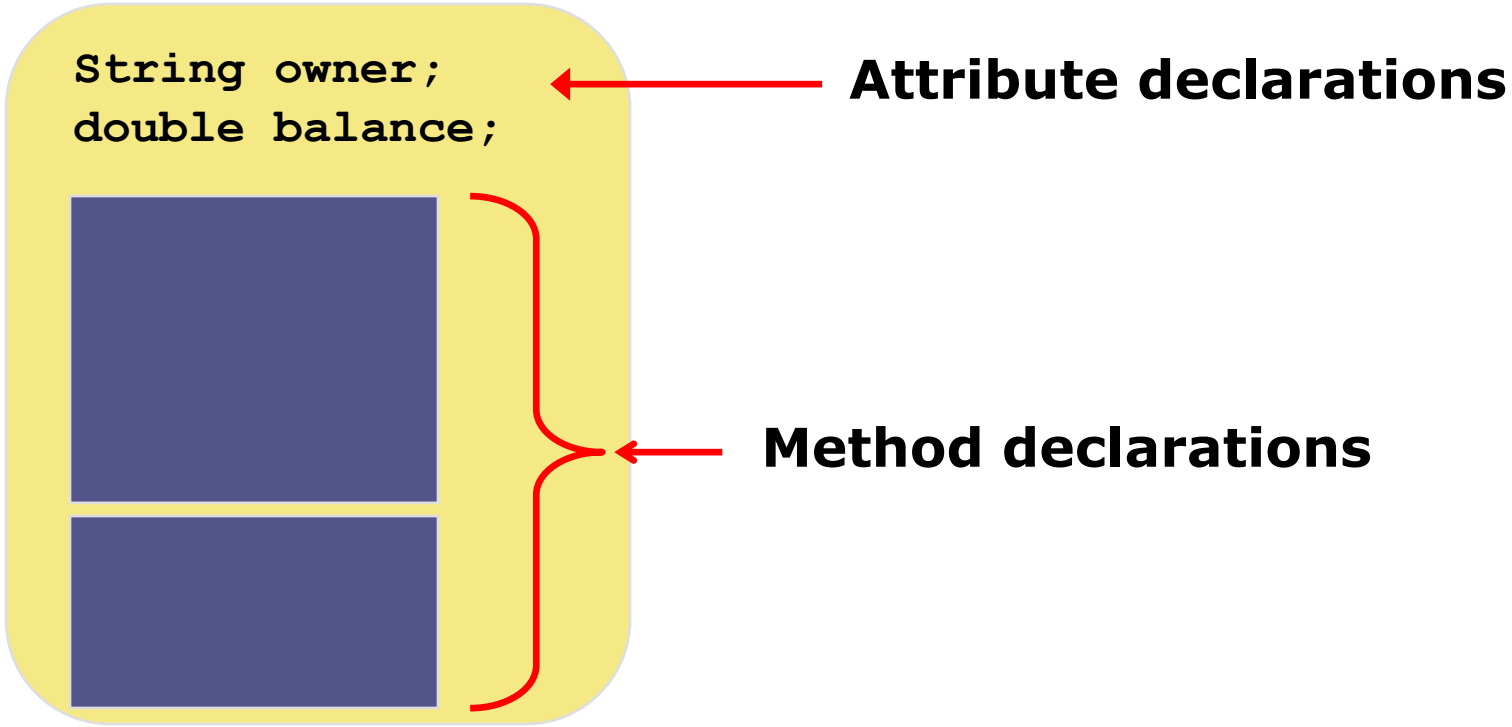
2.2. Class construction

- Necessary information to define a class
  - *The Class Name*
    - Class name describes an object in real life
    - Class name should be singular, short, and clear to the abstraction.
  - The Data Elements
    - The pieces of data that an instance of the class holds are also of vital importance
  - The messages
    - Messages that the object can receive



2.2. Class Construction (2)

- Class encapsulating members
  - Field/attribute
  - Function/method:



C++: Point Class

```
class point {
        double x,y;                    // private data members
    public:
        point (int x0, int y0);          //  public methods
        point () { x = 0; y = 0;};       //  a constructor
         void move (int dx, int dy);
         void rotate (double alpha);
         int distance (point p);
};
```

Rectangle Class

```
class T_hcn {
                int x,y;
                int w,h;
public :
   void set_value(int,int,int,int);
   int area() {
      return (w*h);
   };
} ;
void T_hcn::set_value(int a,in b,int c,int
   d)
{x=a; y=b; w=c; h=d;}
```

```
int main() {
T_hcn hcn;
hcn.set_value(0,0,3
,4);
 cout << ‘’ dien tich
= ‘’ << hcn. area();
 }
```

Method implementation (C++)

- Interace of method is always in the class definition, as well as the declarations of data member.
- The implementation (method definition) can be put inside or outside of the class definition.
- Inside:
  - Inline declaration: method defined inside class declaration
- Outside:
  - Regular declaration: method is defined outside class declaration, and is used with domain operation ::

C++ method

```
class Car {
...
void drive(int speed,
int distance);
};
...
void Car::drive (int speed,
int distance)
{...}
```

```
class Car {
...
void drive(int speed,
int distance) {
  // định nghĩa tại đây
}
};
```

C++: Account class

```
class Account {
public:
  Account();
  float account_balance() const;       // Return the balance
  float withdraw( const float );       // Withdraw from account
  void deposit( const float );         // Deposit into account
  void set_min_balance( const float ); // Set minimum balance
private:
  float the_balance;                   // The outstanding balance
  float the_min_balance;               // The minimum balance
};
```

C++: Account class

```
class Account {
public:
  Account();
  float account_balance() const;       // Return the balance
  float withdraw( const float );       // Withdraw from account
  void deposit( const float );         // Deposit into account
  void set_min_balance( const float ); // Set minimum balance
private:
  float the_balance;                   // The outstanding balance
  float the_min_balance;               // The minimum balance
};
```

C++: Account class

```
Account::Account()
{
  the_balance = the_min_balance = 0.00;
}

float Account::account_balance() const
{
  return the_balance;
}

float Account::withdraw( const float money )
{
  if ( the_balance-money >= the_min_balance
     )
  {
    the_balance = the_balance - money;
    return money;
  } else {    return 0.00;  }
}
```

```
void Account::deposit( const float
money )
{
  the_balance = the_balance +
money;
}

void Account::set_min_balance(
const float money )
{
  the_min_balance = money;
}
```

C++, separation of class decleration and method implemetation

- In general, we should seperate the method declaration from the implementation (definition),
- The separation provides two advantages in encasulating:
  - With the separation from class declaration, users dont need to care about the details (how is a class declaration long and unreadable if it has about 10 to 20 methods, each one has hundreds of lines?)
  - Separation of method interface from implementation allows us to modify the implementation without having impact to users.

Rectangle Class (C++)

```
#include <iostream>
using namespace std;
class Rectangle
{
   private:
      float width, length;
   public:
      void setWidth(float);
      void setLength(float);
      float getWidth(),
        getLength(),
        getArea();
};
```

```
void Rectangle::setWidth(float w)
{width = w;}
void Rectangle::setLength(float
   len)
{length = len;}
float Rectangle::getWidth()
{return width;}
float Rectangle::getLength()
{return length;}
float Rectangle::getArea()
{return width * length;}
```

Class declaration, with prototypes of functions + data

Definition of functions outside the class

Separate declaration (C++)

- To ensure encapsulation, we often declare classes in header file
  - File name is the same as class name. For example, Car class declaration is written in file “car.h”
- The implemetation (definition) is located in a corresponding source file
  - “car.cpp” or “car.cc”
- Convention for declaration/definition of class in a file with the same name is popularly accepted in C++
  - is a required rule in classes of Java

File header Car.h

```
// car.h
#ifndef CAR_H
#define CAR_H
class Car {
    public:
        //...
        void drive(int speed, int distance);
        //...
        void stop();
        //...
        void turnLeft();
    private:
        int vin; //...
        string make; //...
        string model; //...
        string color; //...
};
#endif
```

2.2. Class Construction (3) (Java)

- Classes are grouped into a package
  - Package is composed of a set of classes that have some logic relation between them,
  - Package is considered as a directory, a place to organize classes in order to locate them easily.
- Example:
  - Some packages already available in Java: java.lang, javax.swing, java.io…
  - Package can be manually defined by users
    - Separated by "."
    - Convention for naming package
    - Example: `package oop.k52.cnpm;`

2.2.1. Class declaration

- Declaration syntax:

```
package tenpackage;
access specifier class TenLop {
  // Class body
}
```

  - **`access specifier:`**
    - **`public`**: Class can be accessed from anywhere, including outside its package.
    - **`private`**: Class can only be accessed from inside the class
    - None (default): Class can be accessed from inside its package.

Example – Class declaration

```
package oop.k52.cnpm;

public class Student {
   …
}
```

2.2.2. Member declaration of class

- Class members have access definition similarly to the class.

	`public`	**None**	`private`
Same class			
Same package			
Different package			

2.2.2. Member declaration of class (2)

- Class members have access definition similarly to the class.

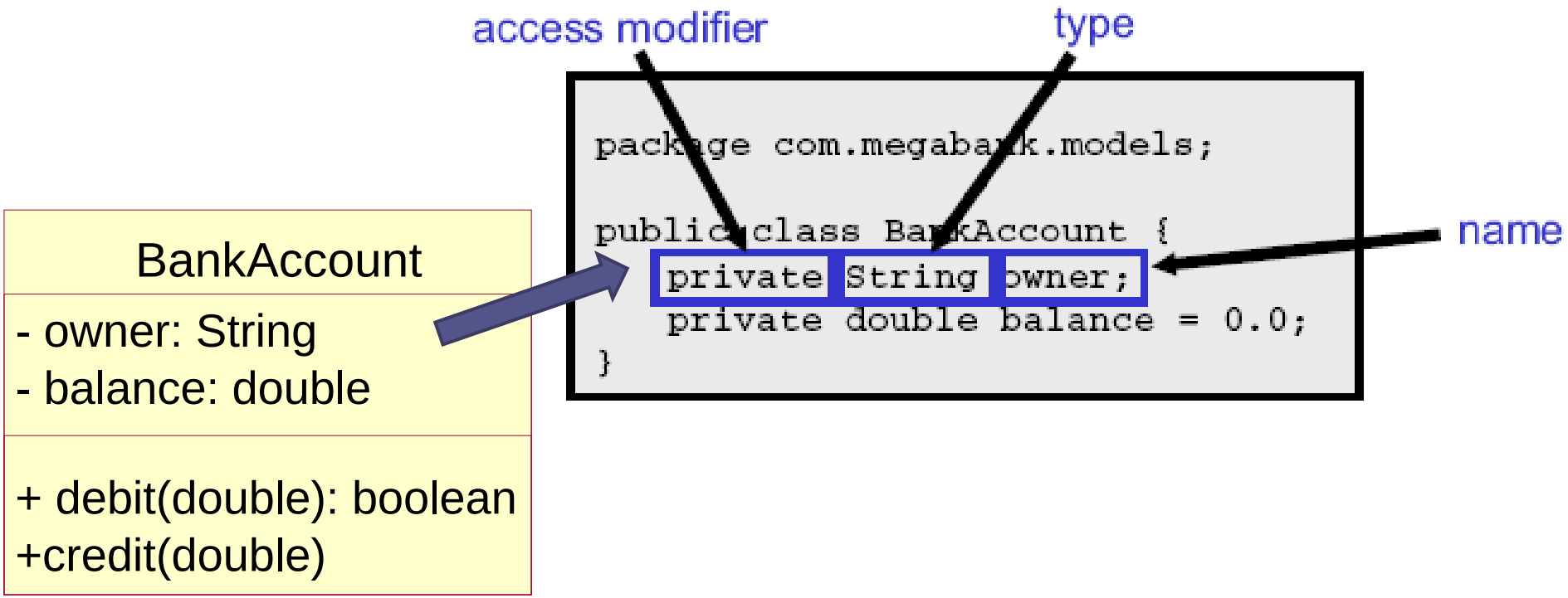
	`public`	**None**	`private`
Same class	Yes	Yes	Yes
Same package	Yes	Yes	No
Different package	Yes	No	No

a. Attribute

- Attributes have to be declared inside the class
- An object has its own copy of attributes
  - The values of an attribute of different objects are different.

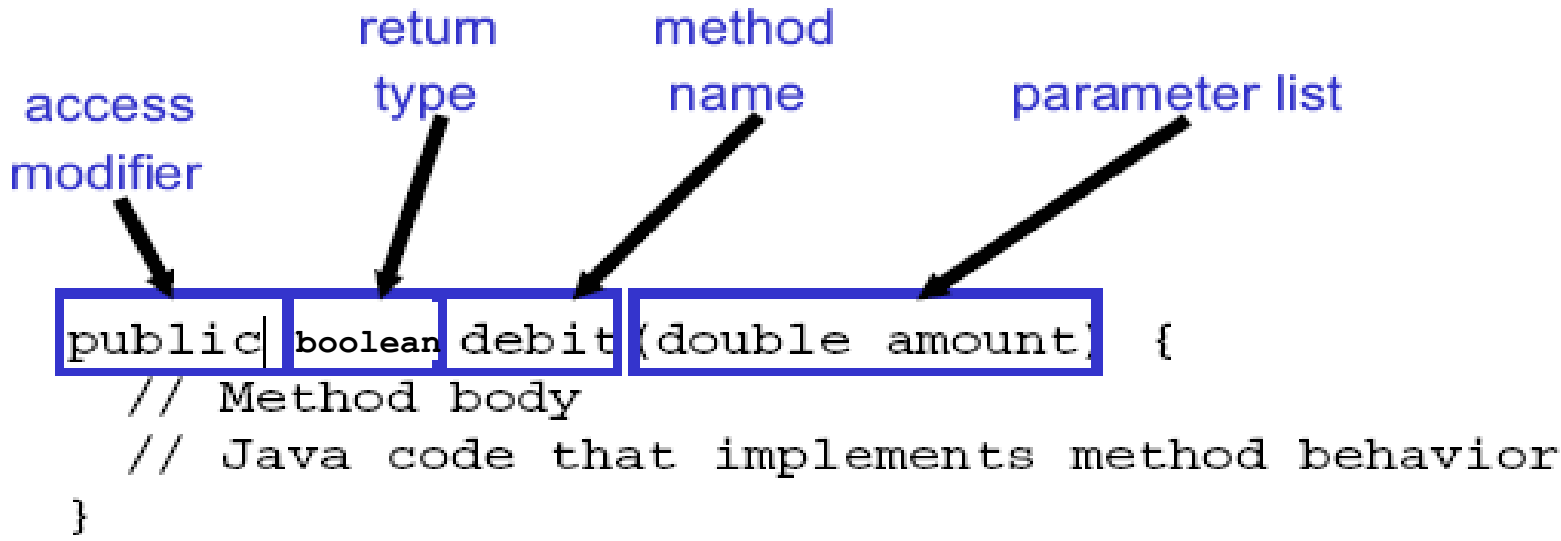
a. Attribute (2)

- Attribute can be initialized while declaring
  - The default value will be used if not initialized.



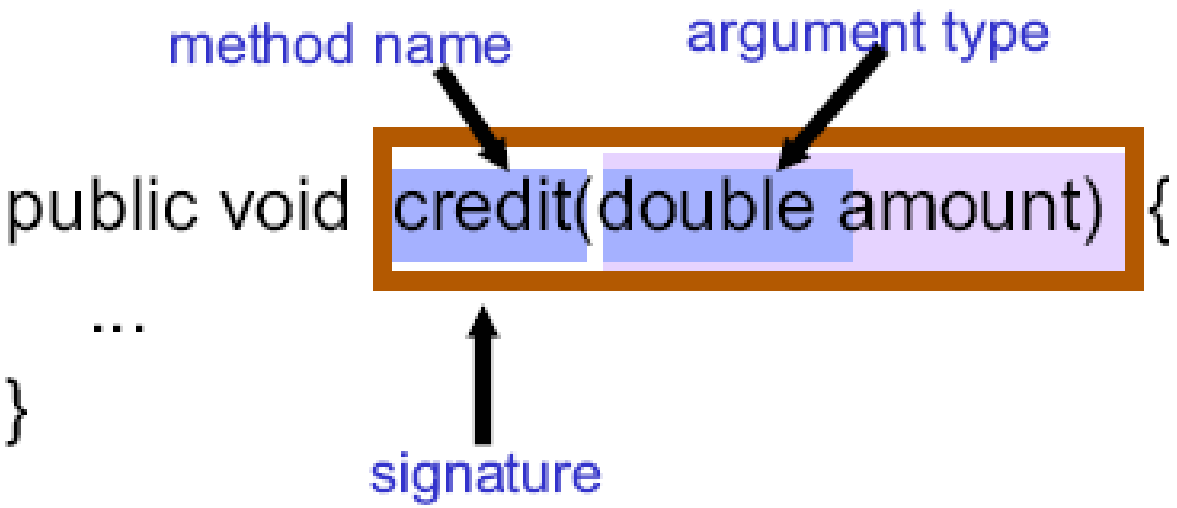
b. Method

- Define how an object responses to a request
- Method specifies the operations of a class
- Any method must belong to a class



* Method signature

- A method has its own signature including:
  - Method name
  - Number of parameters and their types



* Type of returned data

- When a method returns at least a value or an object, there must be a “return” command to return control to the caller object (object that is calling the method).
- If method does not return any value (void), there is no need for the “return” command
- There might be many “return”s in a method; the first one that is reached will be executed.

c. Constant member (Java)

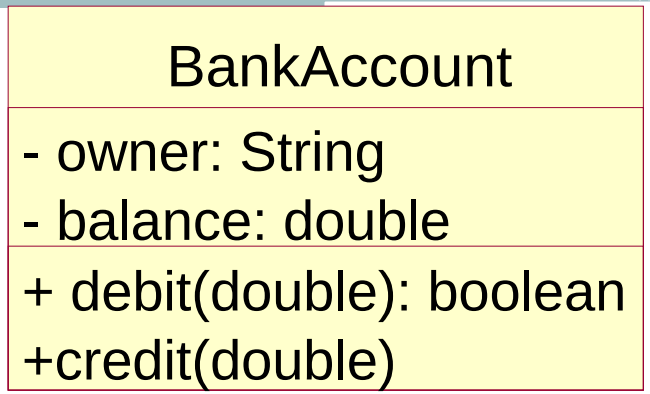
- An attribute/method can not be changed its value during the execution.
- Declaration syntax:

```
chi_dinh_truy_cap final kieu_du_lieu
                  TEN_HANG = gia_tri;
```

- Example:

```
final double PI = 3.141592653589793;
public final int VAL_THREE = 39;
private final int[] A = { 1, 2, 3, 4, 5, 6 };
```

```
package com.megabank.models;
public class BankAccount {
    private String owner;
     private double balance;

     public boolean debit(double amount){
        if (amount > balance)
            return false;
        else {
            balance -= amount; return true;
        }
     }
     public void credit(double amount){
        balance += amount;
     }
}
```



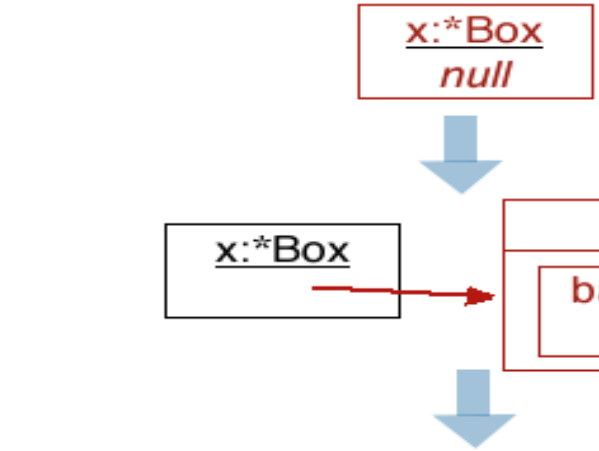
Objects in C++ and Java

- C++: objects of a class are created at the declaration:
  - Point p1;
- Java: The declaration of an object indeed create a reference pointing to the real object created by new operation:
  - Box x;
  - x = new Box();
  - Dynamic memory for objects are allocated in heap memory

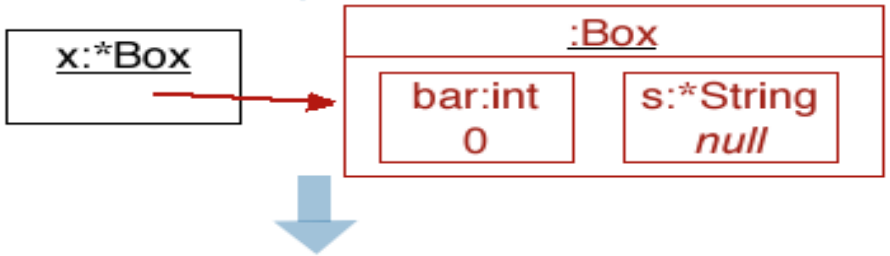
Object in Java

```
class Box
{
    int bar;
    String s;
}
```

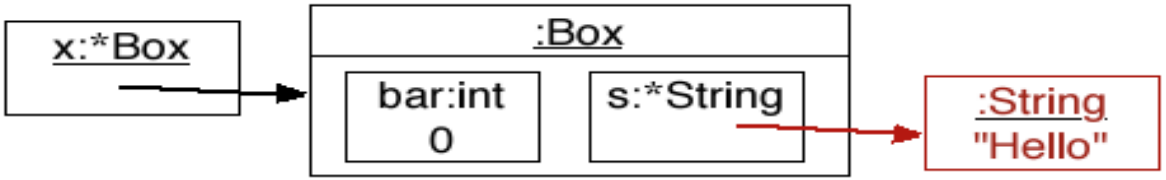
```
Box x;
```




```
x = new Box ();
```




```
x.s = "Hello";
```



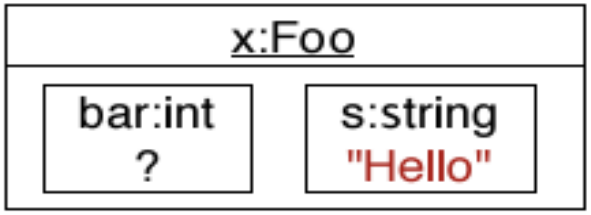
Object in C++

```
class Foo
{
        int bar;
        string s;
};
```

gotta have that semi

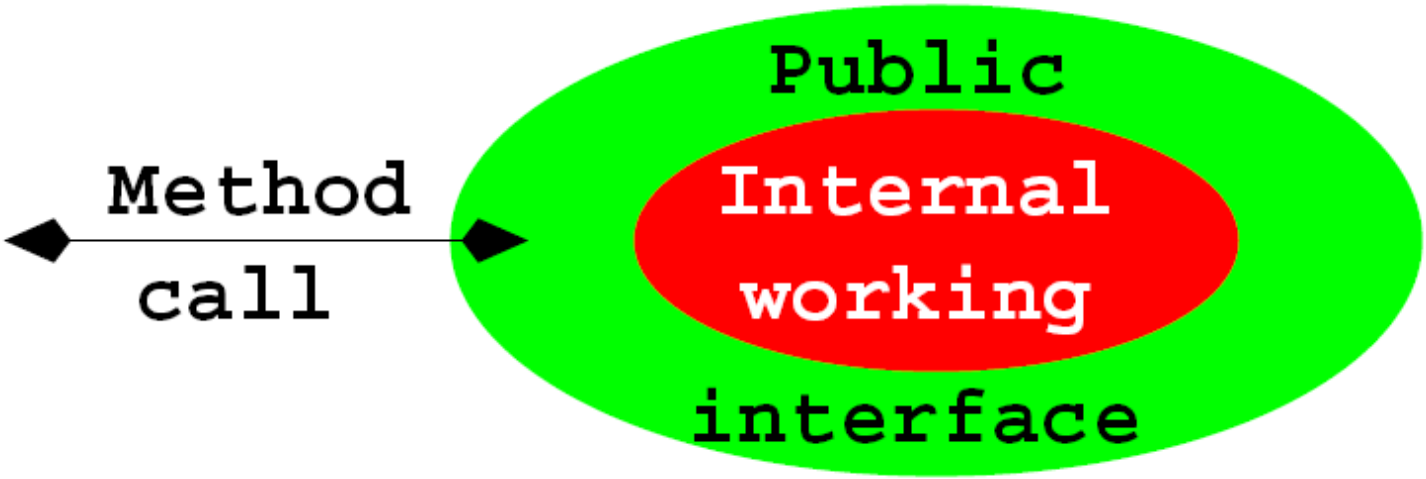
```
Foo x;
```

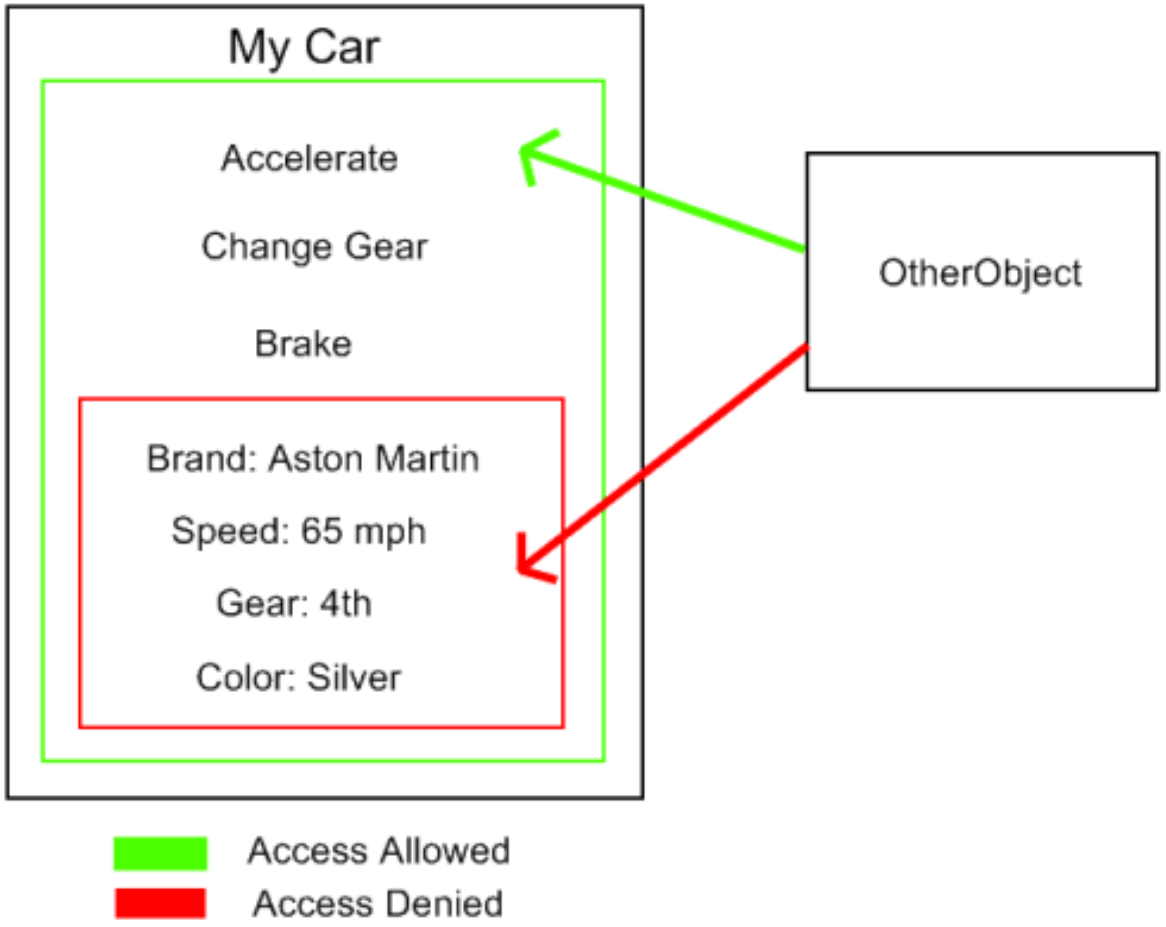

```
x.s = "Hello";
```



2.3. Data hiding

- Data is hidden inside the class and can only be accessed and modified from the methods
  - Avoid illegal modification



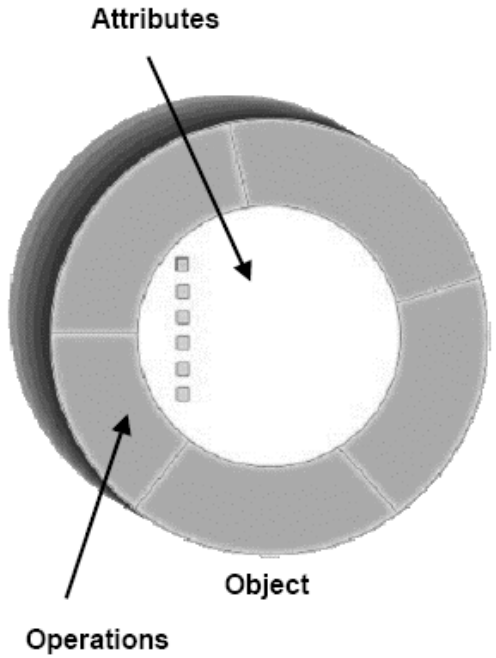

My Car
Accelerate
Change Gear
Brake
Brand: Aston Martin
Speed: 65 mph
Gear: 4th
Color: Silver
OtherObject
Access Allowed
Access Denied

Discussion

- Is the data encapsulation the data hiding?

Data hiding mechanism

- Data member
  - Can only be accessed from methods in the class
  - Access permission is **`private`** in order to protect data
- Other objects that want to access to the private data must perform via public functions



BankAccount
- owner: String - balance: double
+ debit(double): boolean +credit(double)

Data hiding mechanism (2)

- Because data is private → Normally a class provides services to access and modify values of the data
  - Accessor (getter): return the current value of an attribute
  - Mutator (setter): modify value of an attribute
  - Usually getX and setX, where x is attribute name

```
package com.megabank.models;

public class BankAccount {
 private String owner;
 private double balance = 0.0;
}
```

```
public String getOwner() {
  return owner;
}
```

Get Method (Query)

- The Get methods (query method, accessor) are used to get values of data member of an object
- There are several query types:
  - Simple query(*" what is the value of x?"*)
  - Conditional query (*"is x greater than 10?"*)
  - Complex query (*"what is the sum of x and y?"*)
- An important characteristic of getting method is that is should not modify the current state of the object
  - Do not modify the value of any data member.

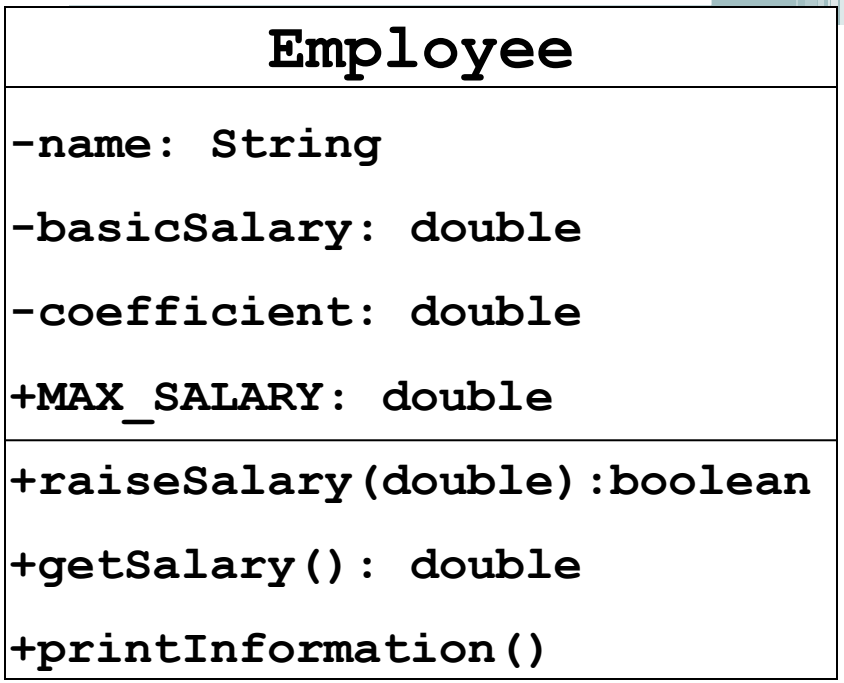
restricted access: *private* members are *not externally accessible*; but we need to know and modify their values

set methods: *public* methods that allow clients to *modify* *private* data; also known as *mutators*

```
public class Time {
    private int hour;
    private int minute;
    private int second;

    public Time () {
        setTime(0, 0, 0);
    }

    public void setHour (int h) { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }

    public void setMinute (int m) { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }

    public void setSecond (int s) { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }

    public void setTime (int h, int m, int s) {
        setHour(h);
        setMinute(m);
        setSecond(s);
    }

    public int getHour () { return hour; }

    public int getMinute () { return minute; }

    public int getSecond () { return second; }
}
```

get methods: *public* methods that allow clients to *read* *private* data; also known as *accessors*

Exercise 1



- Write the code for NhanVien class as in the figure:
  - salary = basic Salary * coefficient
  - Method printInformation () display information of the corresponding NhanVien object.
  - Method **`raiseSalary`** (double) increases the current salary coefficient by the given double input parameter. If this makes the salary > maximal salary then it is not allowed, a notice is printed and function returns false; otherwise it returns true.
- Write methods get and set for the attributes of Employee class.

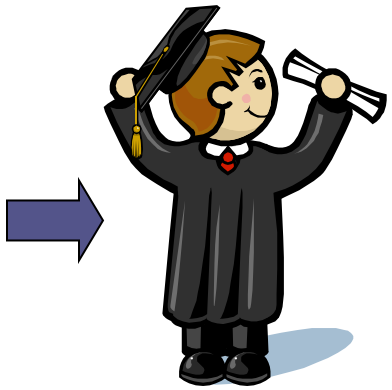
Outline

1. Data abstraction
2. Encapsulation and Class construction
3. Object creation and usage

3.1. Data initialization

- Data need to be initialized before being used
  - Initialization error is one of the most common ones
- For simple/basic data type, use operator =
- For object → Need to use constructor method

Student
- name - address - studentID - dateOfBirth



Nguyễn Hoàng Nam

Hà Nội…

Nguyễn Thu Hương

Hải Phòng…

…

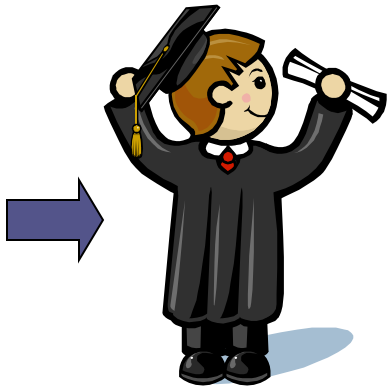
Constrution and destruction of object

- An existing and operating object is allocated some memory by OS in order to store its data values.
- When creating an object, OS will assign initialization values to its attributes
  - Must be done automatically before any developers' operations that are done on the object
  - Using construction function/method
- In contrast, while finishing, we have to release all the memory allocated to objects.
  - Java: JVM
  - C++: destructor

3.2. Constructor method

- Is a particular method that is automatically called when creating an object
- Main goal: Initializing attributes of objects

Student
- name - address - studentID - dateOfBirth



Nguyễn Hoàng Nam

Hà Nội…

Nguyễn Thu Hương

Hải Phòng…

…

3.2. Constructor method(2)

- Every class must have at least one constructor
  - To create a new representation of the class
  - Constructor name is the same as the class name
  - Constructor does not have return data type
- For example:

```
public BankAccount(String o, double b){
   owner = o;
   balance = b;
}
```

3.2. Constructor method (3)

- Constructor can use access modifier:
  - **`public`**
  - **`private`**
  - None (default – can be used in package)
- A constructor can not use the keywords **`abstract`**, **`static`**, **`final`**, **`native`**, **`synchronized`**.
- Constructors can not be considered as *class members*.

3.2. Constructor method (4)

- Default constructor
  - Is a constructor **without parameters**

```
public BankAccount(){
   owner = “noname”; balance = 100000;
}
```

  - If we do not write any constructor in a class
    - New JVM provides a default constructor
    - The default constructor provided by JVM has the same access attributes as its class
  - A class should have a default constructor

3.3. Object declaration and initialization

- An object is created and instantiated from a class.
- Objects have to be declared with **Types of objects** before being used:
  - Object type is object class
  - For example:
    - `String strName;`
    - `BankAccount acc;`

3.3. Object declaration and initialization (2)

- Objects must be initialized before being used
  - Use the operator = to assign
  - Use the keyword **`new`** for constructor to initialize objects:
    - Keyword **`new`** is used to create a new object
    - Automatically call the corresponding constructor
  - The default initialization of an object is **`null`**
- An object is manipulated through its *reference (~ pointer)*.
- For example:

```
BankAccount acc1;
acc1 = new BankAccount();
```

3.3. Object declaration and initialization (3)

- We can combine the declaration and the initialization of objects
  - Syntax:

```
Class_name object_name = new
                         Constructor(parameters);
```

  - For example:

```
BankAccount account = new BankAccount();
```

3.3. Object declaration and initialization (4)

- Constructor does not have **return value**, but when being used with the keyword **new,** it returns a reference pointing to the new object.

```
public BankAccount(String name) {
   setOwner(name);
}
```

Constructor definition

Constructor use

```
BankAccount account = new BankAccount("Joe Smith");
```

3.3. Object declaration and initialization (5)

- Array of objects is declared similarly to the array of primitive data
- Array of objects is initialized with the value **`null`.**
- For example:

```
Employee emp1 = new Employee(123456);
Employee emp2;
emp2 = emp1;
Department dept[] = new Department[100];
Test[] t = {new Test(1),new Test(2)};
```

Example 1

```
class BankAccount{
  private String owner;
  private double balance;
}
public class Test{
  public static void main(String args[]){
      BankAccount acc1 = new BankAccount();
  }
}
```

→ Default constructor provided by Java.

Example 2

```
public class BackAccount{
  private String owner;
  private double balance;
  public BankAccount(){
    owner = ”noname”;
  }
}
public class Test{
  public static void main(String args[]){
      BankAccount acc1 = new BankAccount();
  }
}
```

→ Default constructor written by developers.

Example 3

```
public class BankAccount {
  private String owner;
  private double balance;
  public BankAccount(String name){
    setOwner(name);
  }
  public void setOwner(String o){
    owner = o;
  }
}
public class Test{
  public static void main(String args[]){
   BankAccount account1 = new BankAccount(); //Error
    BankAccount account2 = new BankAccount("Hoang");
  }
}
```

Constructor with implicit arguments (C++)

```
class Automobile {
public:
  Automobile();

  Automobile( string make, int doors,
      int cylinders = 4, int engineSize =
   2000 );

  Automobile( const Automobile & A );
  // copy constructor
```

Copy constructor

- The goal of copy function is to create a new object and copy data from an existing object to the newly created object.
- The syntax of copy function is as follows:

  <Name> (<type> &) ;

  or <Name> (const <type> &) ;
- Keyword const in the parameters is to prevent all the changes of paramenters' values.
- We can create a new object that have several similar attributes compared to an existing object, not all the attributes are the same like the assignment.

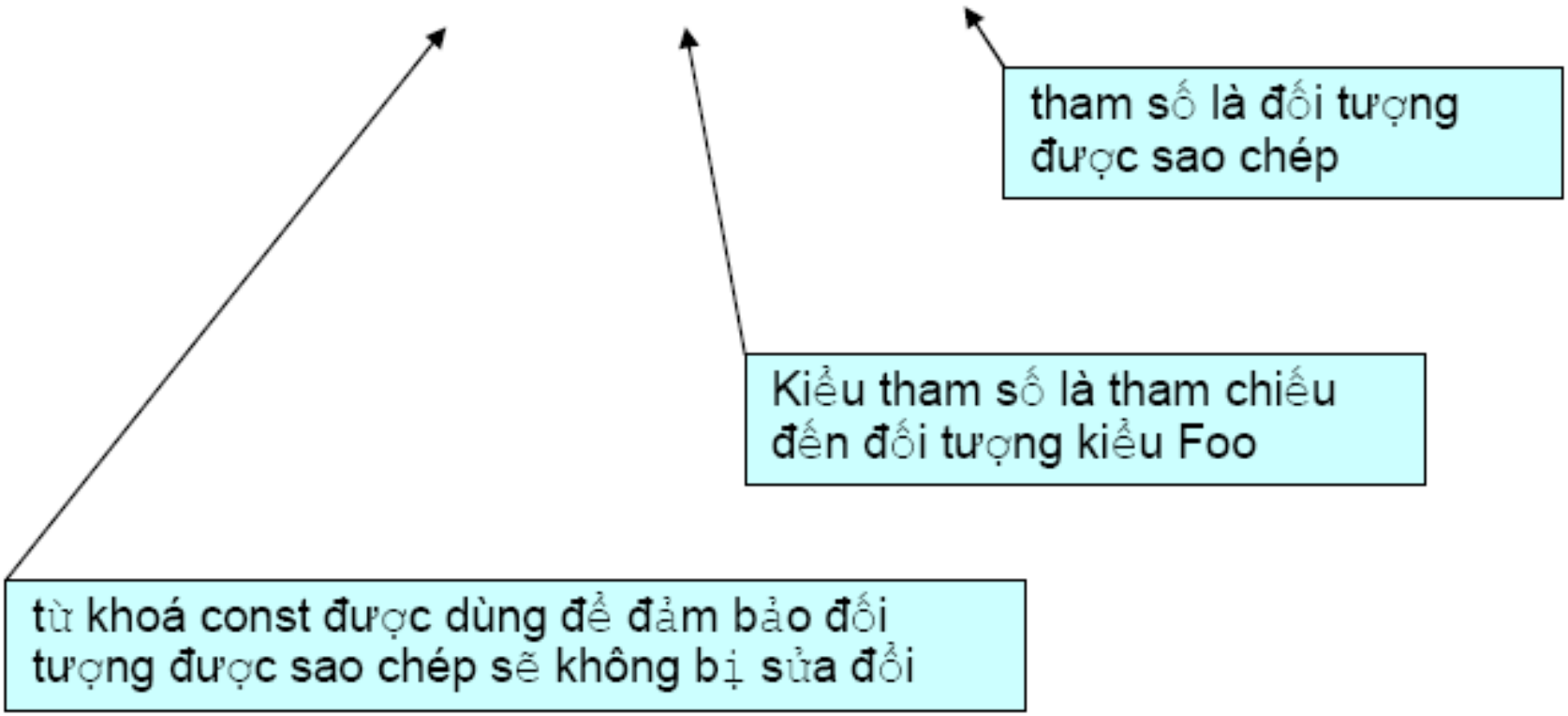
Copy constructor

```
MyClass x(5);
MyClass y = x; hoặc MyClass y(x);
```

- C++ provides a copy constructor, it simply copies every attribute from an object to a new object.
- However, in many cases we need to do some other initialization activities in copy constructor
  - For example: get a value of a unique ID from somewhere, or perform a “deep” copy (for example when a member is a pointer for dynamic allocator)
- In these cases, we can re-define the copy constructor

Typical declaration

```
Foo(const Foo& existingFoo);
```



```
class Person {
    public:
    Person(const char *name0="", int
    age0=0);
            Person(const Person &p);
            void print();
    private:
            char name[30];
            int age;
};
```

Sử dụng tường minh hàm thiết lập sao chép:

```
Person person("Matti", 20);
Person twinBrother(person);
```

```
Person::Person(const
    Person &p) {
    strcpy(name, p.name);
    age = p.age;
}
```

Sử dụng không tường minh:

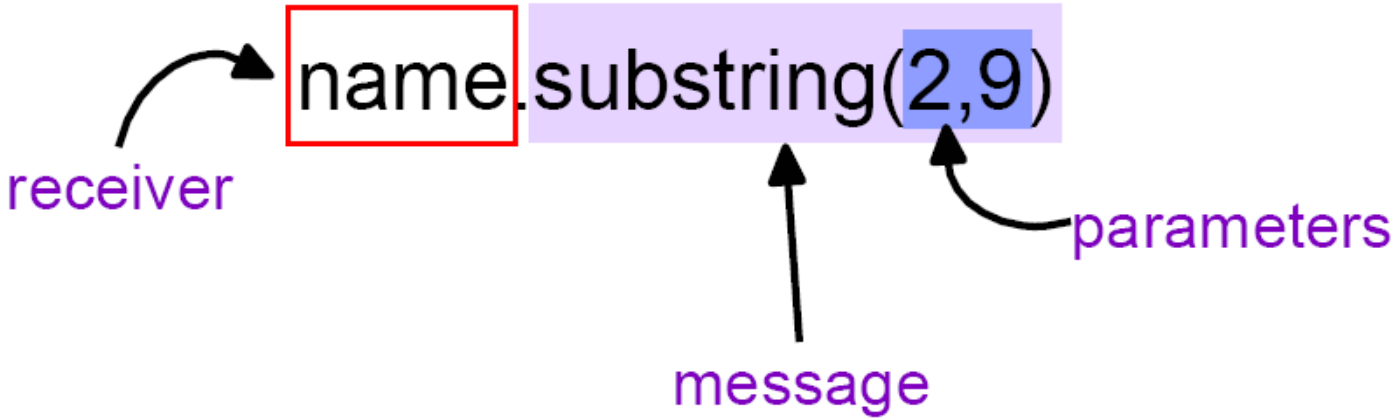
```
void f(Person p);
void main(void)   {
    Person person(“Matti”, 20);
    f(person);
}
```

Copy constructor

- Notice on memory leaking when writing code for copy constructor
- In Java, there is no concept of copy constructor.

3.4. Object usage

- Object provides more complex operations than primitive data types.
- Objects responds to messages
  - Operator "." is used to send a message to an object



3.4. Object usage (2)

- To call a member (data or attribute) of a class or of an object, we use the operator “.”
- If we call method right in the class, the operator “.” is not necessary.

```
BankAccount account = new BankAccount();
account.setOwner("Smith");
account.credit(1000.0);
System.out.println(account.getBalance());
...
```

BankAccount method

```
public void credit(double amount) {
    setBalance(getBalance() + amount);
}
```

```
public class BankAccount{
  private String owner;
  private double balance;
  public BankAccount(String name) {
    setOwner(name);
  }
  public void setOwner(String o){ owner = o; }
  public String getOwner(){ return owner; }
}
public class Test{
  public static void main(String args[]){
   BankAccount acc1 = new BankAccount("");
   BankAccount acc2 = new BankAccount("Hong");
   acc1.setOwner("Hoa");
   System.out.println(acc1.getOwner()
                      + " "+ acc2.getOwner());
}
```

Destructor (C++)

- In contrast to Constructor, when releasing an object we have to release all the memory allocated for the object. Destructor function is for this purpose:
- Example: class A {

```
int n;
public:
  A();  //constructor
 ~A();  // destructor
};
```

- Java: there is no destructor.

Destructor

- Before OS releases the memory allocated to the storage of object data, the destructor is executed. Hence, in the destructor, we only need to release what the OS can not automatically release.

- Principles to notice:
  - Classes contains some attributes that are pointers
  - Do not forget to release memory (the memory usage is not efficient) and do not free memories that are allocated twice (will have errors)

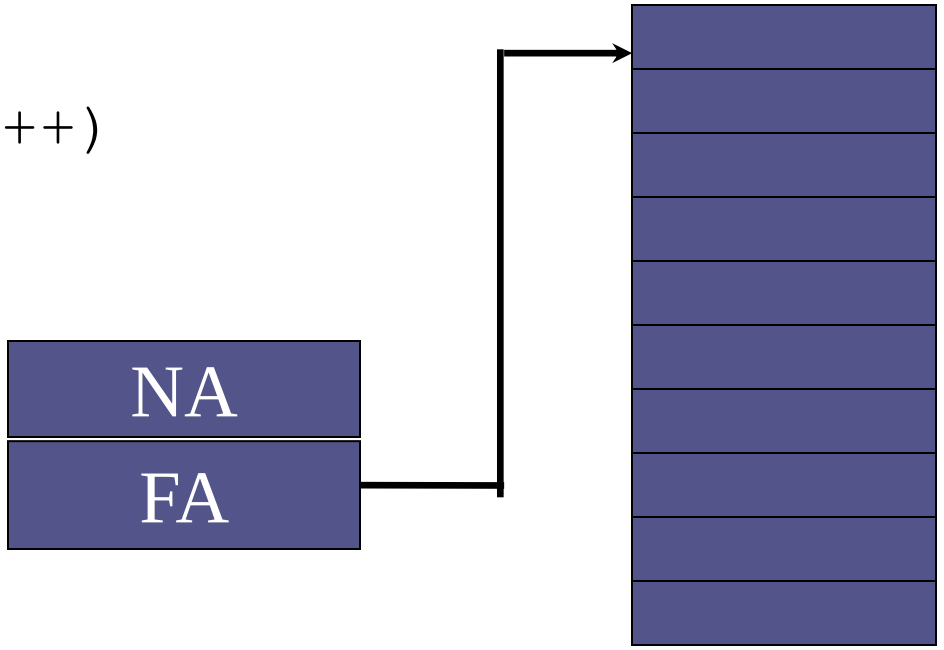
Object destructor

```
#include <iostream>
#include <conio.h>
#include <stdio.h>

class A
 { int NA;
   float *FA;
   public:
   A(int m);
   void display();
 };
```

```
A::A(int m)
{
 NA=m;
 FA = new float [m];
 for (int i=0; i<m; i++)
  { FA[i]=i*10.0; }
 }
```

Object destructor

```
void A::display()   {
for (int i=0; i<NA; i++)
      { cout << FA[i];
        cout << "   ";
      }
}

void main()
{  A A1(5);
   A1.display();
}
```



Release A1,
Array still exists?
Need to completely release the memory?

Destructor function

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

class A
 { int NA;
   float *FA;
   public:
   A(int m);
   void display();
   ~A();
 };
```

```
A::A(int m)
 {
  NA=m;
  FA = new float [m];
  for (int i=0; i<m;
  i++)
    { FA[i]=i*10.0; }
  }
A::~A()
 { delete [] FA;}
```

Self-reference- this

- Allows to access to the current object of class.
- Is important when function/method is operating on two or many objects.
- Removes the mis-understanding between a local variable, parameters and data attributes of class.
- Is not used in static code block

C++

```
int point::coincide(point pt)    {
  return(this->x==pt.x && this->y==pt.y);
}

void point::display()  {
   cout<<"Dia chi : "<<this<<"Toa do :
  "<<x<<" "<<y<<"\n";
}
```

Java

```
public class Person
{
...
public void setName(String name)
{
this.name = name;
}
...
private String name;
private int age;
}// end class Person
```

```
public class BankAccount{
  private String owner;
  private double balance;
  public BankAccount() { }
  public void setOwner(String owner){
    this.owner = owner;
  }
  public String getOwner(){ return owner; }
}
public class Test{
  public static void main(String args[]){
    BankAccount acc1 = new BankAccount();
    BankAccount acc2 = new BankAccount();
    acc1.setOwner("Hoa");
    acc2.setOwner("Hong");
    System.out.println(acc1.getOwner() + " " +
                       acc2.getOwner());
}
```

Exercise 2

NhanVien
-tenNhanVien: String -luongCoBan: double -heSoLuong: double +LUONG_MAX: double
+tangLuong(double):boolean +tinhLuong(): double +inTTin()

- Write code for NhanVien class (already done)
- Write constructor methods with neccessary parameters to initialize the attributes of NhanVien class.
- Write class TestNV where 2 objects of NhanVien class are created, and pass message to every newly created object in order to display information, display salary, salary increasing,...

Exercise: Black Jack

- This is a warm-up exercise to refresh your knowledge of objects and classes in a
- concrete way. This exercise is to be done in a small group. You will play a game of
- Blackjack and identify the key classes, objects and interactions.
- The rules of basic Blackjack are given below – read through them.
- Nominate one person as the dealer. Deal out two cards to each player.
- Play the game a few times, and then discuss the following:
- o What are the key classes?
- o What are the important characteristics of each class - that is, what information
- do you need to know about each class? Write down a list of characteristics for
- each class.
- o What are the objects that exist for this particular game? Give each object a
- name and decide which type of class it is.
- o What objects interact with each other?
- You should finish with a set of classes, objects and interactions between objects.