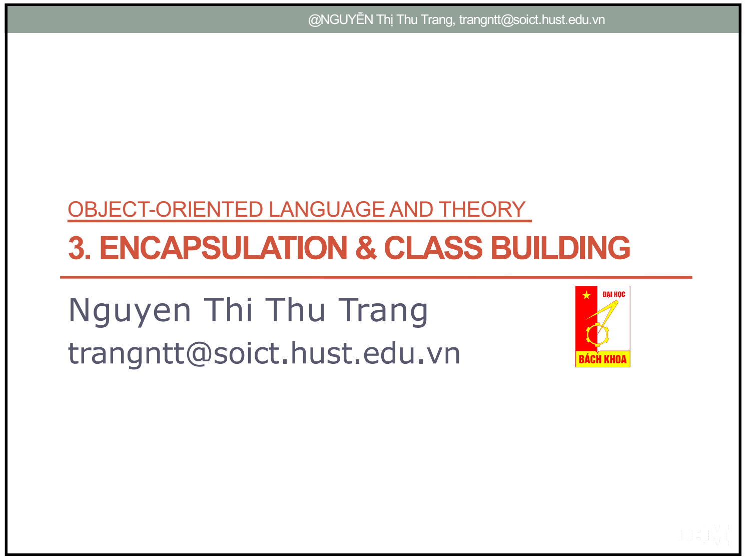
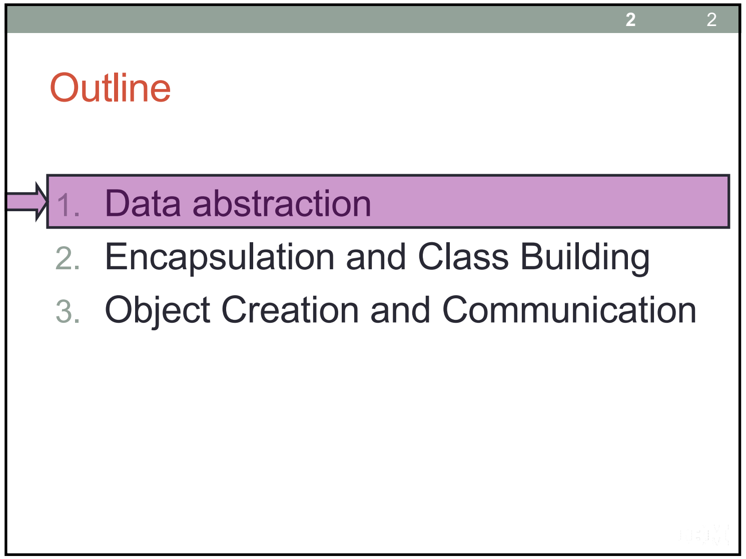


OBJECT-ORIENTED LANGUAGE AND THEORY

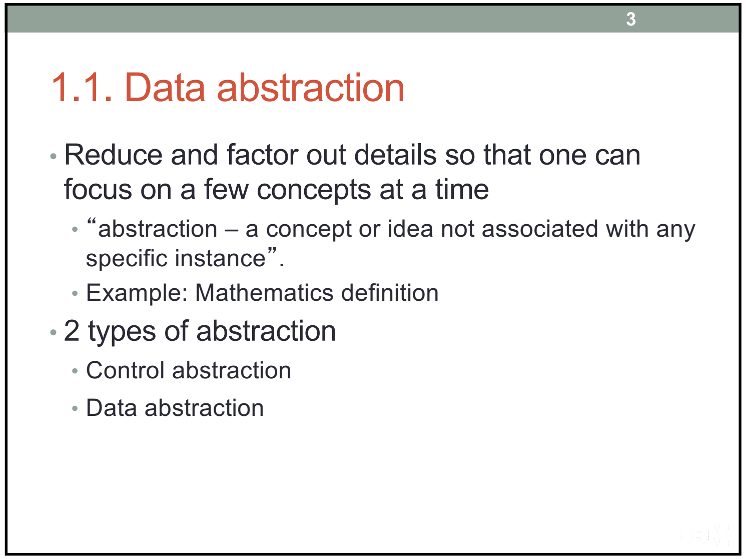
3. ENCAPSULATION & CLASS BUILDING

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn



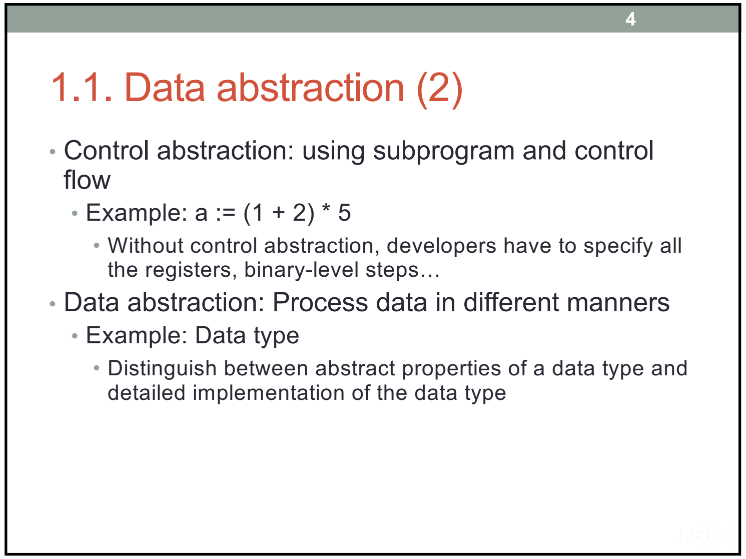
Outline

1. Data abstraction
2. Encapsulation and Class Building
3. Object Creation and Communication



1.1. Data abstraction

- 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
- 2 types of abstraction
  - Control abstraction
  - Data abstraction



1.1. Data abstraction (2)

- Control abstraction: using subprogram and control flow
  - Example: a := (1 + 2) * 5
    - Without control abstraction, developers have to specify all the registers, binary-level steps…
- Data abstraction: Process data in different manners
  - Example: Data type
    - Distinguish between abstract properties of a data type and detailed implementation of the data type

Example of Abstraction

GURU99.COM

1.2. Data abstraction in OOP

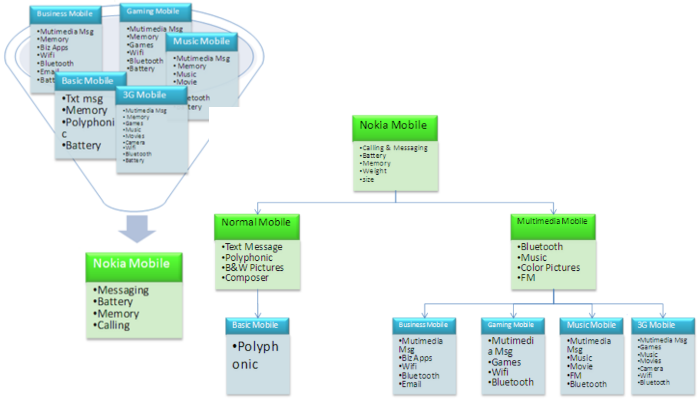
- Objects in reality are very complex

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

Example: Abstract Nokia phones

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

From details to generalization



Business Mobile
•Mutimedia Msg
•Memory
•Biz Apps
•Wifi
•Bluetooth
•Email
•Batt

Gaming Mobile
•Mutimedia Msg
•Memory
•Games
•Wifi
•Bluetooth
•Battery

Music Mobile
•Mutimedia Msg
•Memory
•Music
•Movie
etooth
tery

Basic Mobile
•Txt msg
•Memory
•Polyphonic
•Battery

3G Mobile
•Mutimedia Msg
•Memory
•Games
•Music
•Movies
•Camera
•Wifi
•Bluetooth
•Battery

Nokia Mobile
•Messaging
•Battery
•Memory
•Calling

Nokia Mobile
•Calling & Messaging
•Battery
•Memory
•Weight
•size

Normal Mobile
•Text Message
•Polyphonic
•B&W Pictures
•Composer

Multimedia Mobile
•Bluetooth
•Music
•Color Pictures
•FM

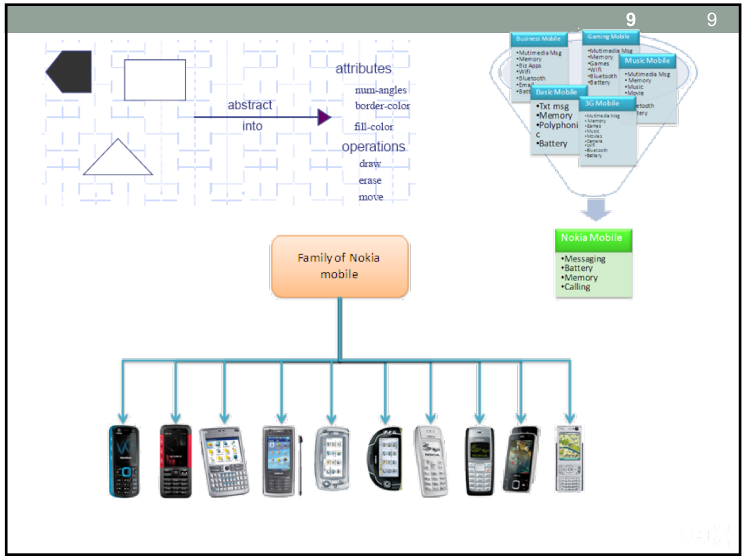
Basic Mobile
•Polyphonic

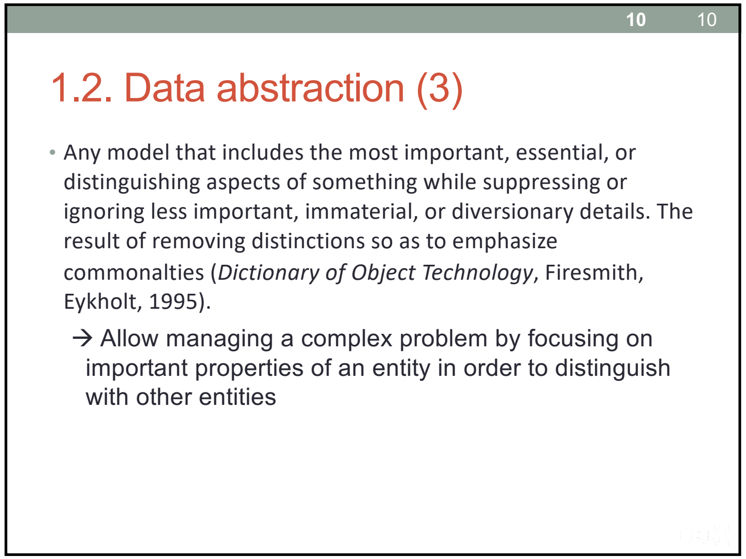
Business Mobile
•Mutimedia Msg
•Biz Apps
•Wifi
•Bluetooth
•Email

Gaming Mobile
•Mutimedia Msg
•Games
•Wifi
•Bluetooth

Music Mobile
•Mutimedia Msg
•Music
•Movie
•FM
•Bluetooth

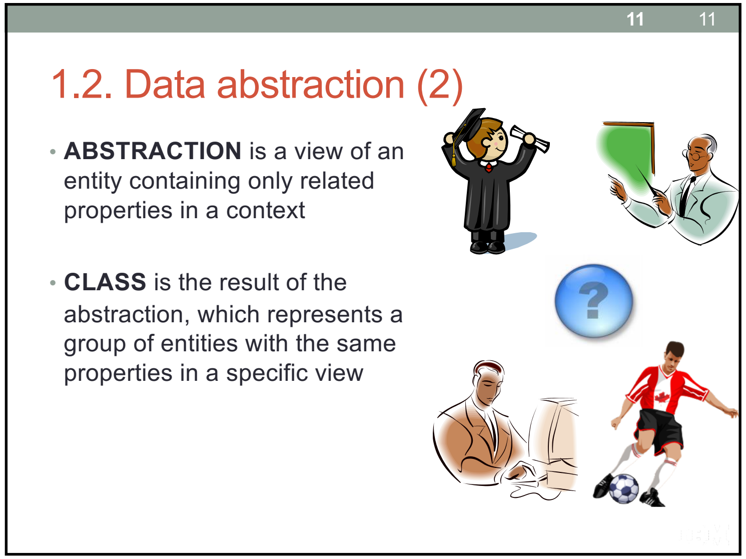
3G Mobile
•Mutimedia Msg
•Games
•Music
•Movies
•Camera
•Wifi
•Bluetooth





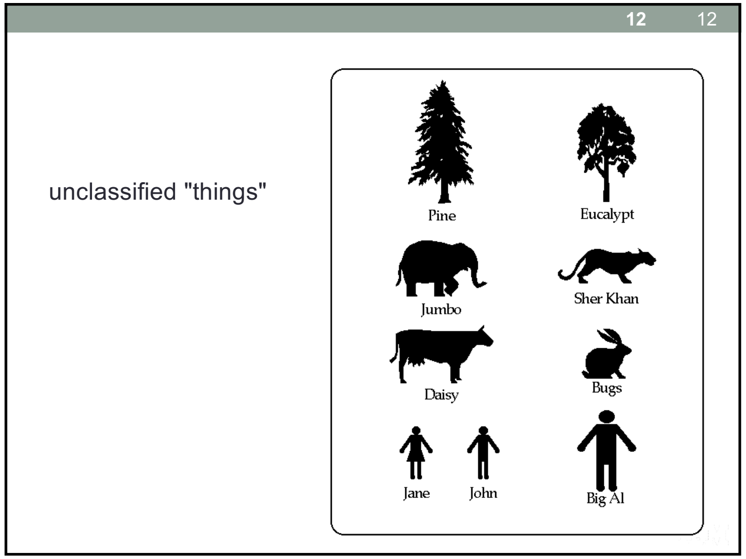
1.2. Data abstraction (3)

- 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).
  - → Allow managing a complex problem by focusing on important properties of an entity in order to distinguish with other entities



1.2. Data abstraction (2)

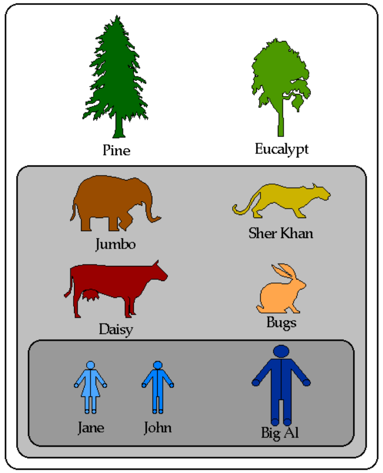
- **ABSTRACTION** is a view of an entity containing only related properties in a context
- **CLASS** is the result of the abstraction, which represents a group of entities with the same properties in a specific view



unclassified "things"

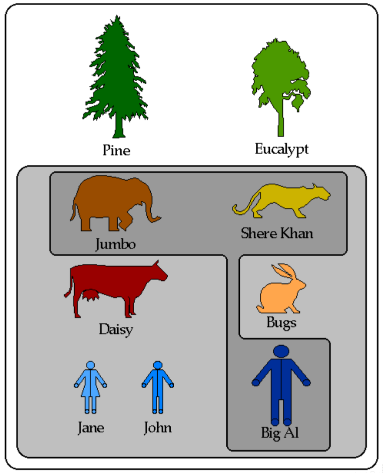
13

- organisms, mammals, humans



14

- organisms, mammals, dangerous mammals



15

1.3. Class vs. Objects

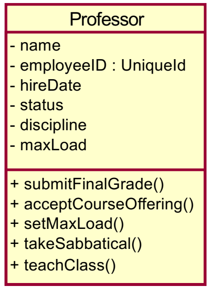
- Class is concept model, describing entities
- Class is a prototype/ blueprint, defining common properties and methods of objects
- A class is an abstraction of a set of objects.

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

16

Class representation in UML

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



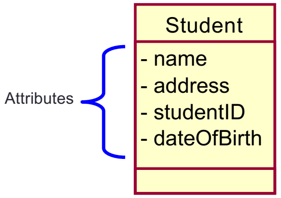
Professor

- name
- employeeID : UniqueId
- hireDate
- status
- discipline
- maxLoad

+ submitFinalGrade()
+ acceptCourseOffering()
+ setMaxLoad()
+ takeSabbatical()
+ teachClass()

What is attribute?

- 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.



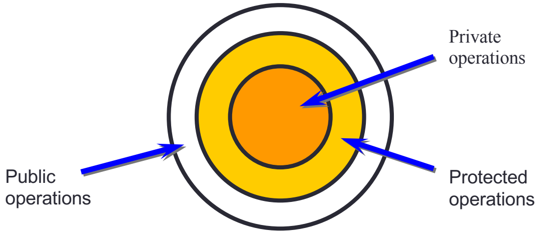
Attributes

Student

- name
- address
- studentID
- dateOfBirth

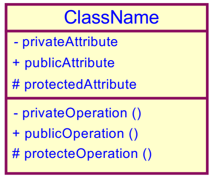
Operation Visibility

- Visibility is used to enforce encapsulation
- May be public, protected, or private



How Is Visibility Noted?

- The following symbols are used to specify export control:
  - + Public access
  - # Protected access
  - - Private access

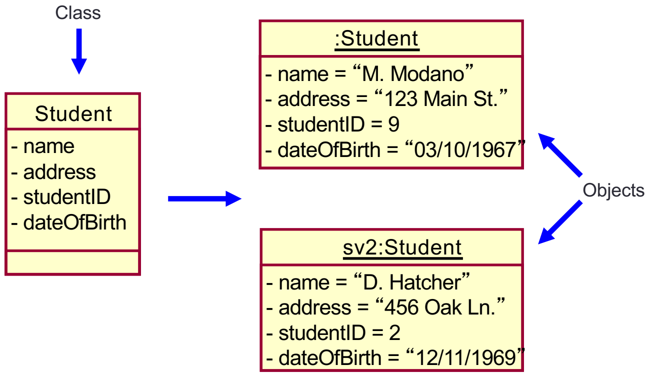


ClassName

- privateAttribute
+ publicAttribute
# protectedAttribute

- privateOperation ()
+ publicOperation ()
# protecteOperation ()

Class and Object in UML



Class

Student

- name
- address
- studentID
- dateOfBirth

:Student

- name = “M. Modano”
- address = “123 Main St.”
- studentID = 9
- dateOfBirth = “03/10/1967”

Objects

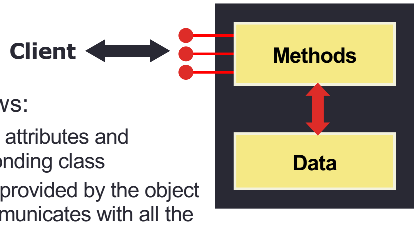
sv2:Student

- name = “D. Hatcher”
- address = “456 Oak Ln.”
- studentID = 2
- dateOfBirth = “12/11/1969”

Outline

1. Data abstraction
2. Encapsulation and Class Building
3. Object Creation and Communication

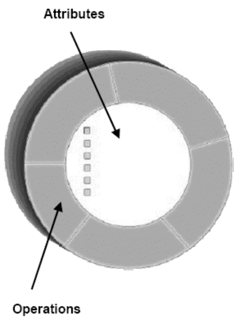
2.1. Encapsulation



- 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)

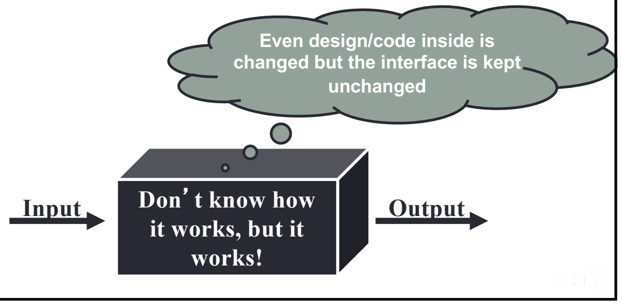
- Data/properties and behavior/methods are encapsulated in a class → Encapsulation



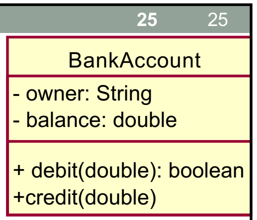
BankAccount
- owner: String - balance: double
+ debit(double): boolean + credit(double)

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


Even design/code inside is changed but the interface is kept unchanged
Input
Don’ t know how it works, but it works!
Output

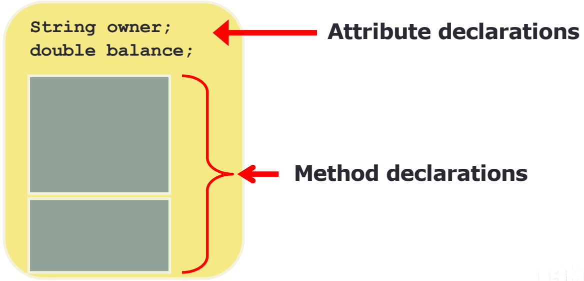
2.2. Class Building



- **Class name**
  - Specify what the abstraction is capturing
  - Should be singular, short, and clear identify the concept
- **Data elements**
  - The pieces of data that an instance of the class holds
- **Operations/Messages**
  - List of messages that instances can receive
- **Methods**
  - Implementations of the messages that each instance can receive

2.2. Class Building (2)

- Class encapsulating members
  - Attributes
  - Methods

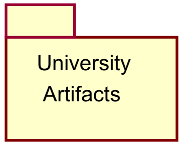


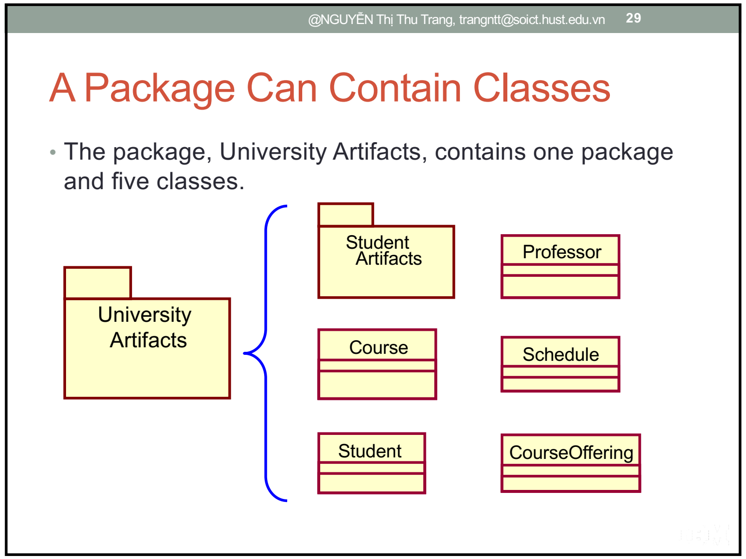
Class Building in Java

- 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`…
  - Packages can be manually defined by users
    - Separated by “.”
    - Convention for naming package
    - Example: `package oolt.hedspi;`

Package in UML

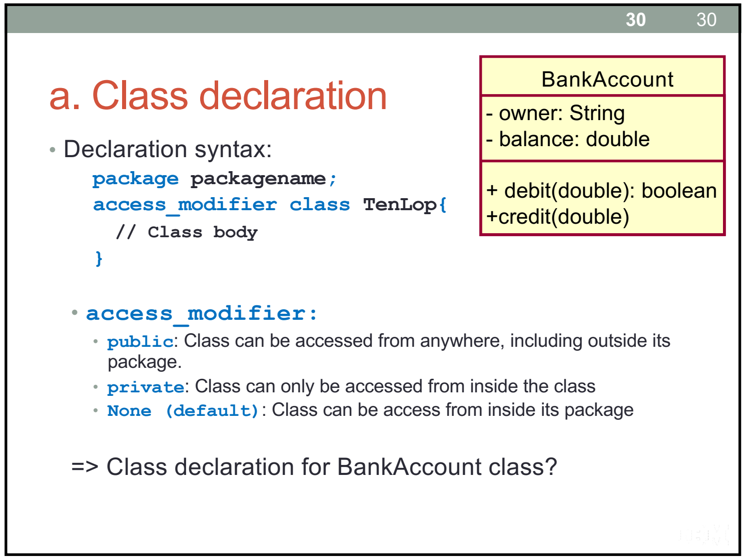
- A general purpose mechanism for organizing elements into groups.
- A model element that can contain other model elements.
- A package can be used:
  - To organize the model under development
  - As a unit of configuration management





A Package Can Contain Classes

- The package, University Artifacts, contains one package and five classes.



a. Class declaration

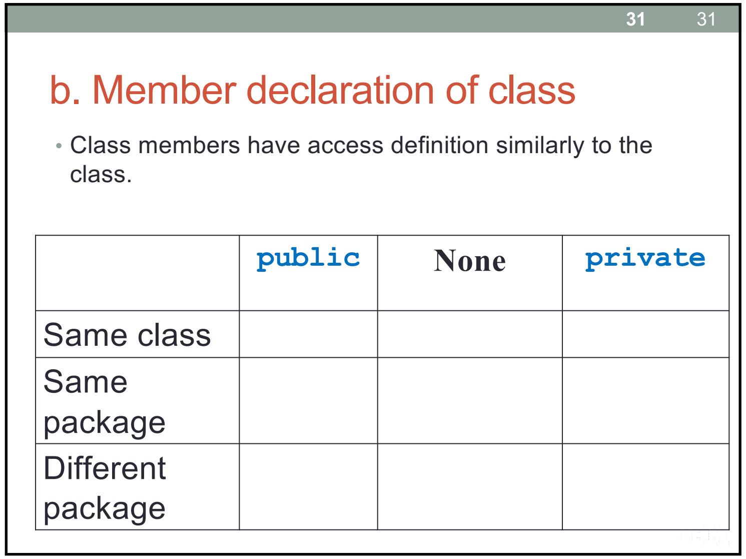
- Declaration syntax:

```
package packagename;
access_modifier class TenLop{
  // Class body
}
```

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

- `access_modifier:`
  - `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 access from inside its package

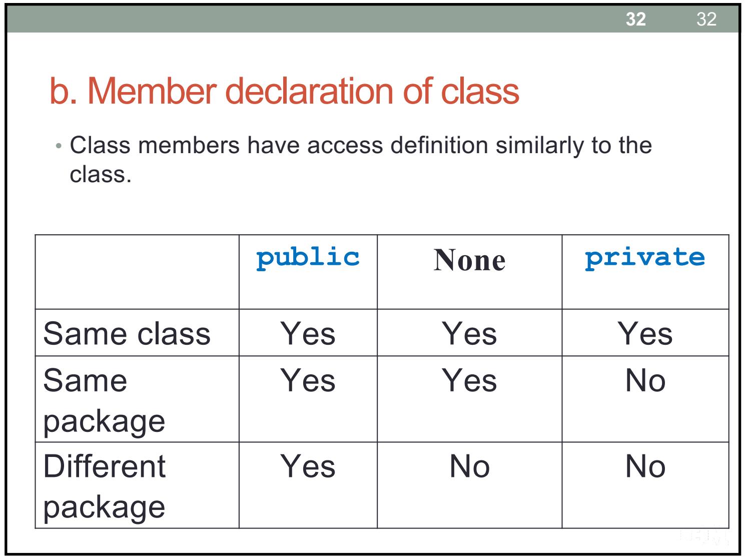
=> Class declaration for BankAccount class?



b. Member declaration of class

- Class members have access definition similarly to the class.

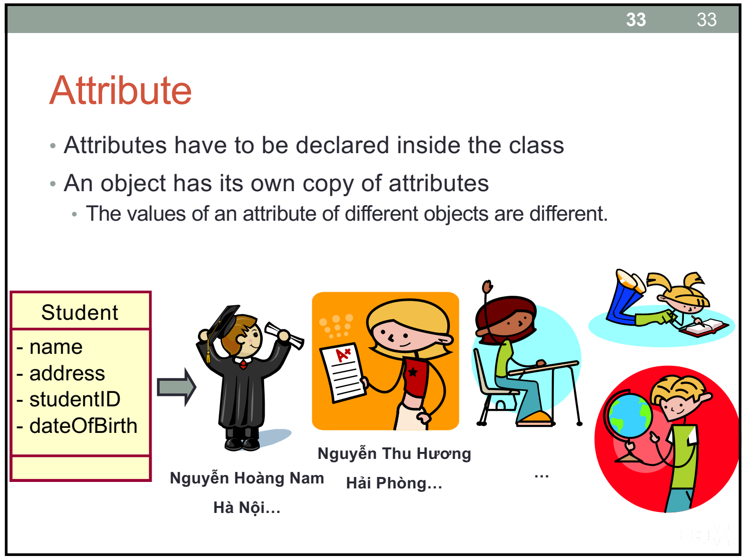
	public	None	private
Same class			
Same package			
Different package			



b. Member declaration of class

- 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

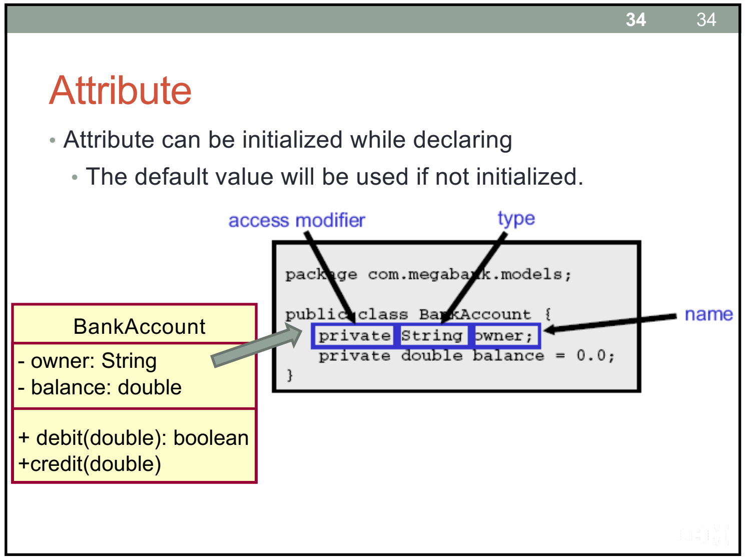


33 33

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.

Student
- name - address - studentID - dateOfBirth



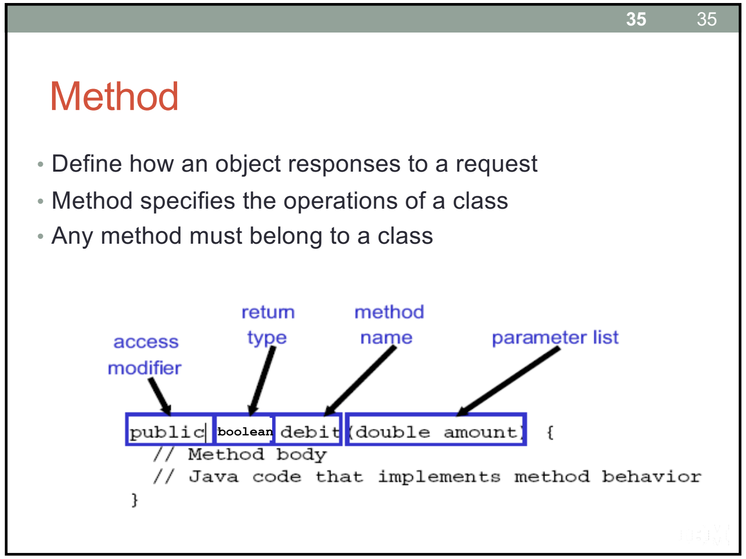
34 34

Attribute

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

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

```
package com.megabank.models;

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

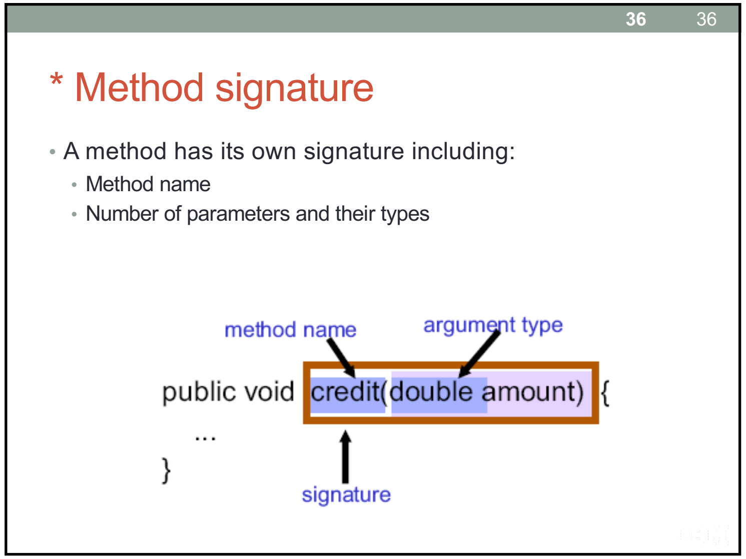


35 35

Method

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

```
public boolean debit(double amount) {
   // Method body
   // Java code that implements method behavior
}
```



36 36

* Method signature

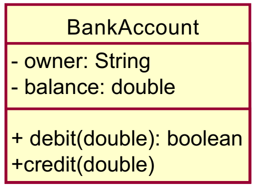
- A method has its own signature including:
  - Method name
  - Number of parameters and their types

```
public void credit(double amount) {
   ...
}
```

* 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.

Class Building Example



- Example of a private field
  – Only this class can access the field

```
balance private double balance;
```

- Example of a public accessor method
  - Other classes can ask what the balance is

```
public double getBalance() {
      return balance;
}
```

- Other classes can change the balance only by calling deposit or withdraw methods

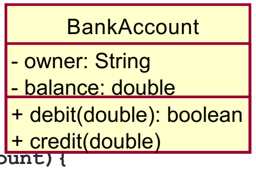
c. Constant member (Java)

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

```
access_modifier final data_type
        CONSTANT_NAME = value;
```

- Example:

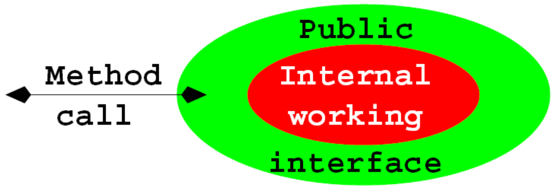
```
final double PI = 3.141592653589793;
public final int VAL_THREE = 39;
private final int[] A = { 1, 2, 3, 4, 5, 6 };
```



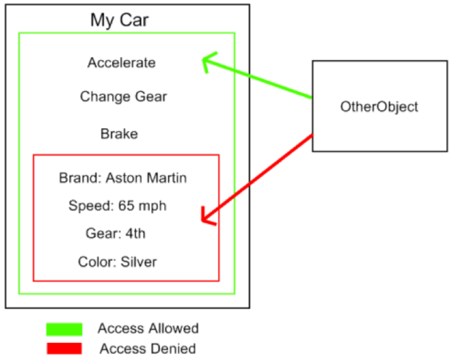
```
package com.megabank.models;
public class BankAccount {
   private String owner;
    private double balance;

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

2.3. Data hiding

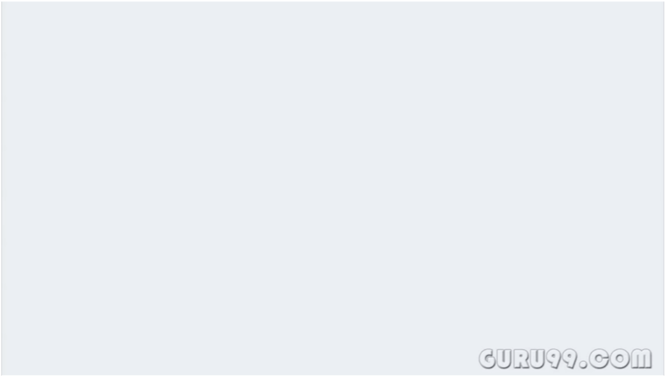
- Data is hidden inside the class and can only be accessed and modified from the methods
  - Avoid illegal modification



Example – Data hiding

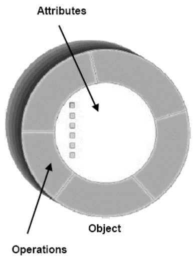


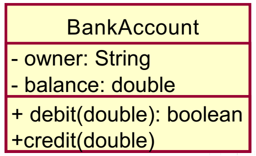
Encapsulation with Java



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

```
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; }
}
```

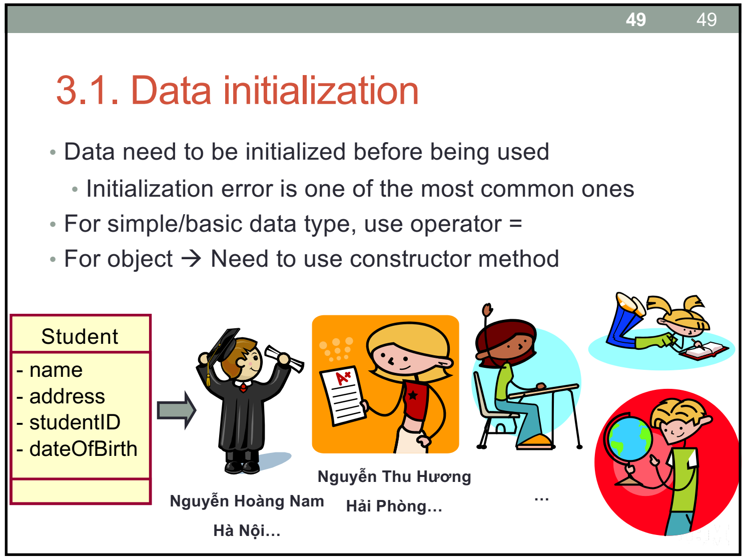
restricted access: *private* members are *not externally accessible*; but we need to know and modify their values

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

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

Outline

1. Data abstraction
2. Encapsulation and Class Building
3. Object Creation and Communication



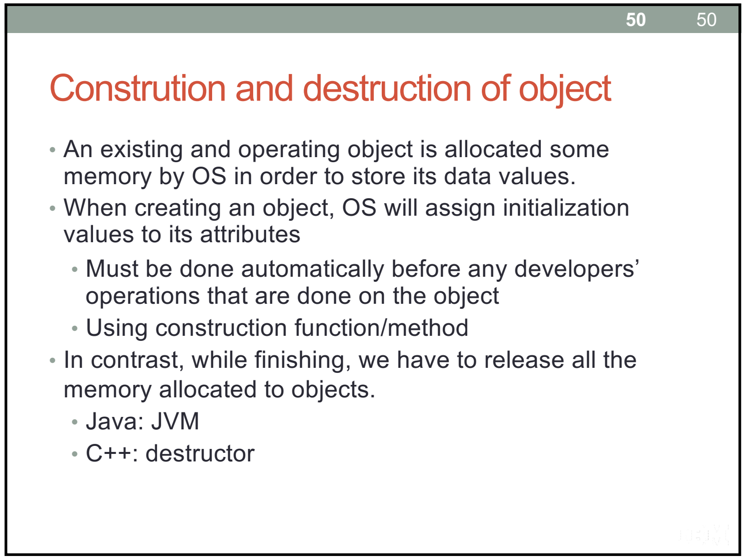
3.1. Data initialization

- 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

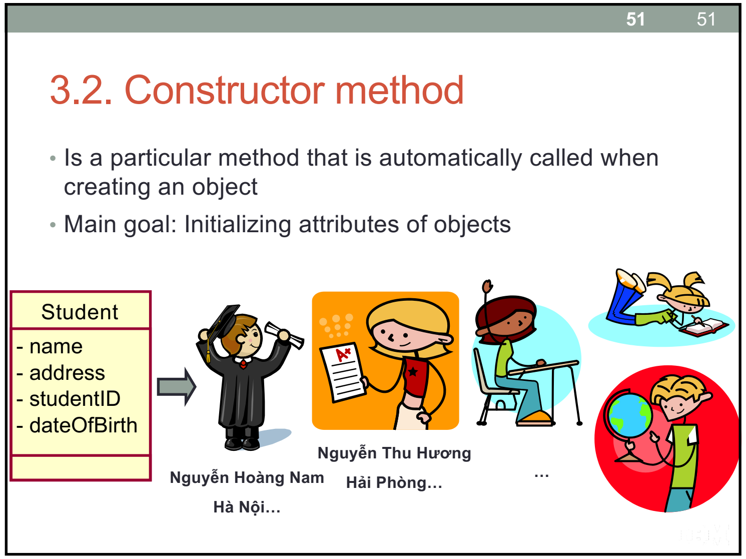
- name
- address
- studentID
- dateOfBirth

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



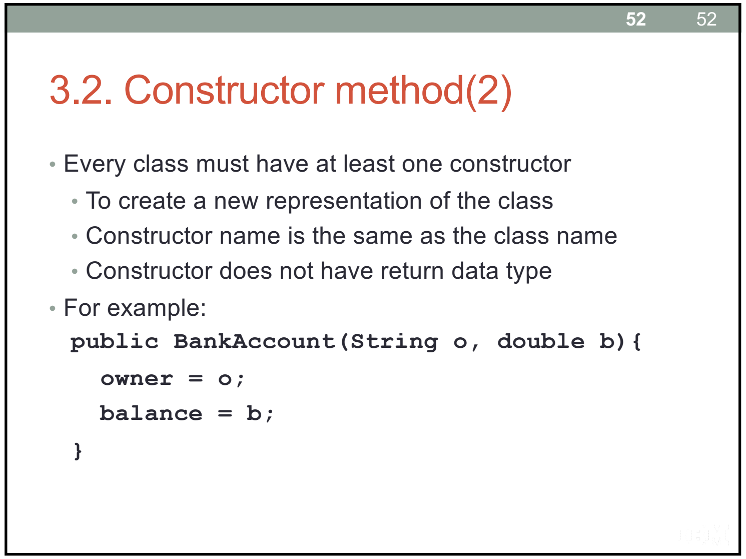
3.2. Constructor method

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

Student

- name
- address
- studentID
- dateOfBirth

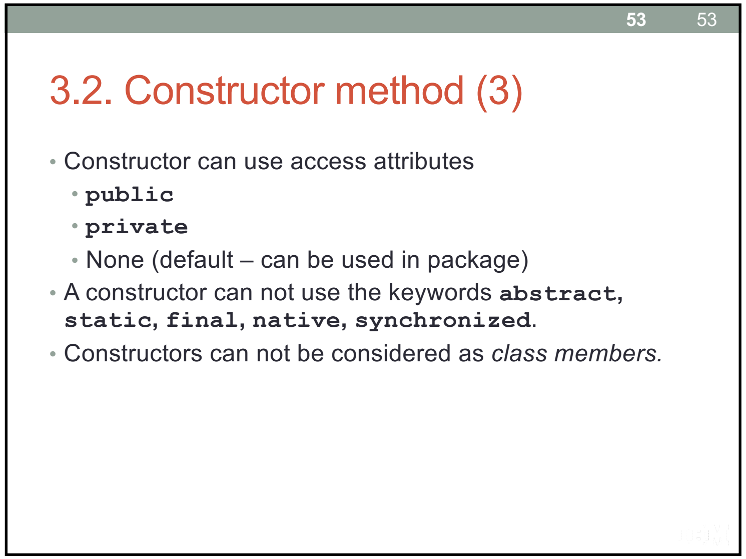
Nguyễn Hoàng Nam
Hà Nội...
Nguyễn Thu Hương
Hải Phòng...
...



3.2. Constructor method(2)

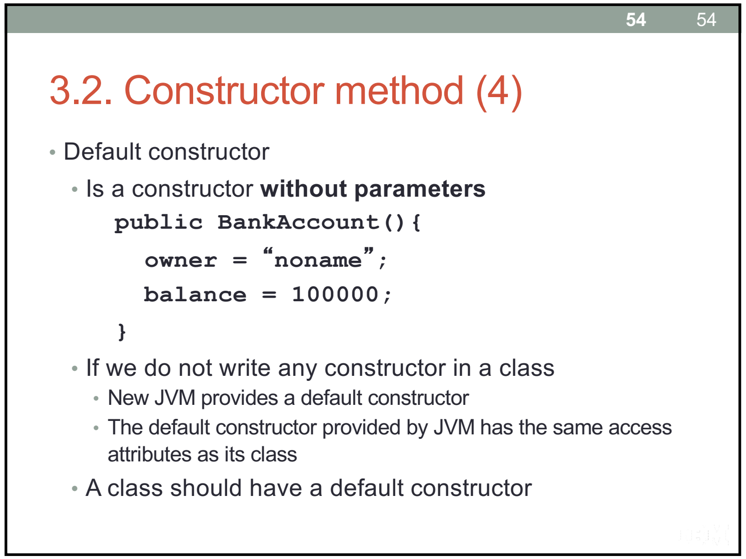
- Every class must have at least one constructor
  - To create a new representation of the class
  - Constructor name is the same as the class name
  - Constructor does not have return data type
- For example:

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



3.2. Constructor method (3)

- Constructor can use access attributes
  - **`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.*

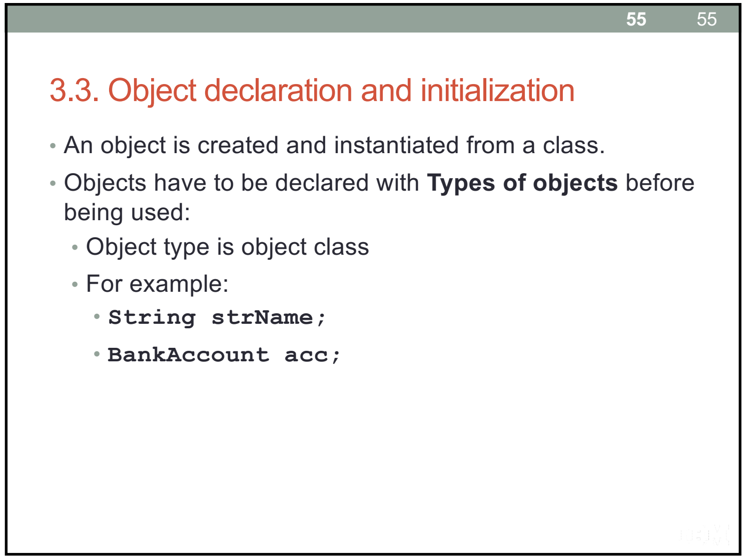


3.2. Constructor method (4)

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

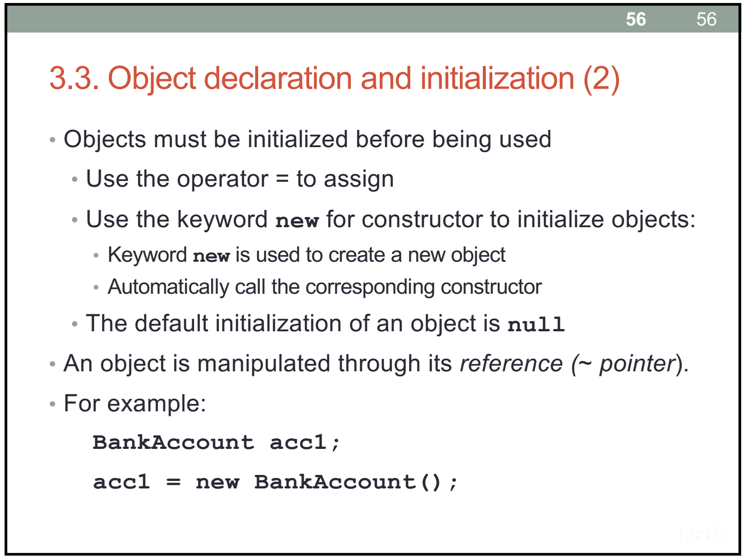
```
public BankAccount(){
   owner = “noname”;
   balance = 100000;
}
```

  - 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

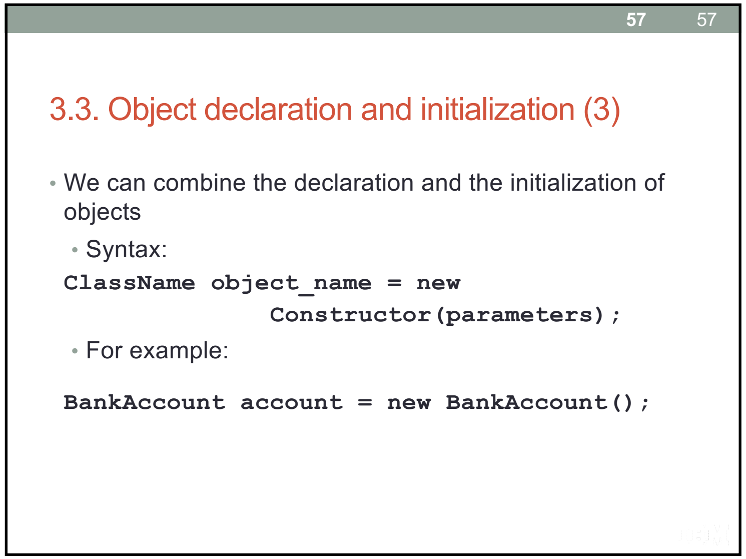
- An object is created and instantiated from a class.
- Objects have to be declared with **Types of objects** before being used:
  - Object type is object class
  - For example:
    - **`String strName;`**
    - **`BankAccount acc;`**



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();
```



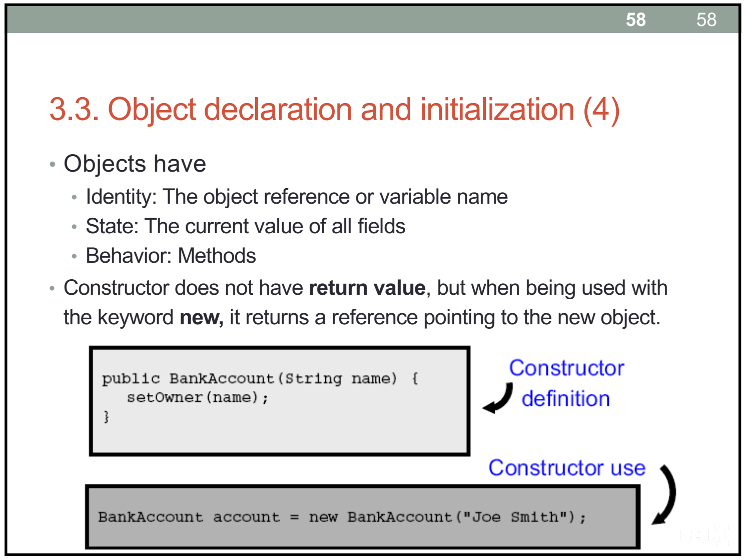
3.3. Object declaration and initialization (3)

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

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

  - For example:

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



3.3. Object declaration and initialization (4)

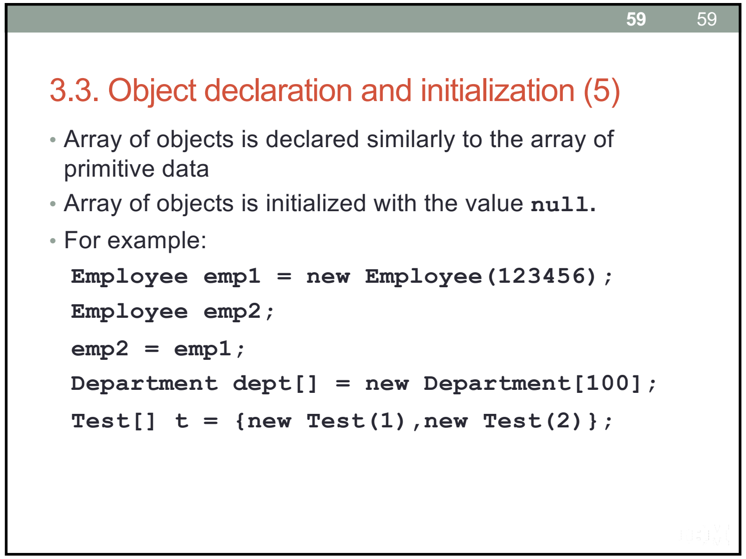
- Objects have
  - Identity: The object reference or variable name
  - State: The current value of all fields
  - Behavior: Methods
- Constructor does not have **return value**, but when being used with the keyword **new,** it returns a reference pointing to the new object.

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

Constructor definition

Constructor use

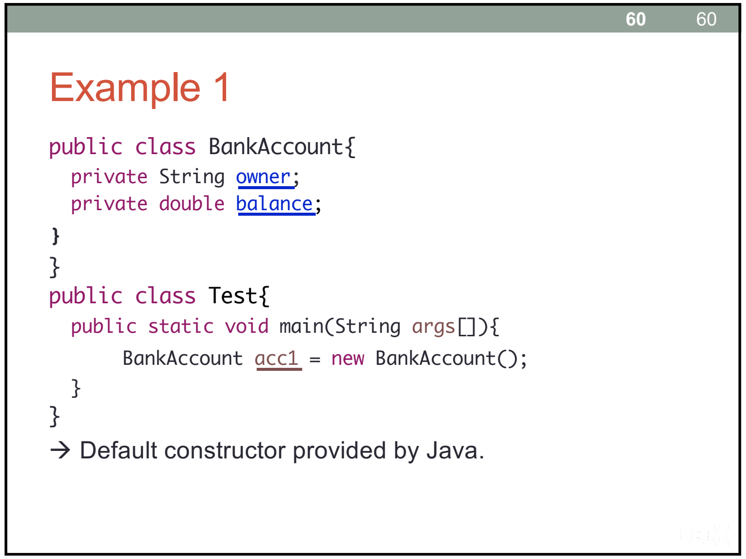
```
BankAccount account = new BankAccount("Joe Smith");
```



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

```
public 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 BankAccount{
  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.

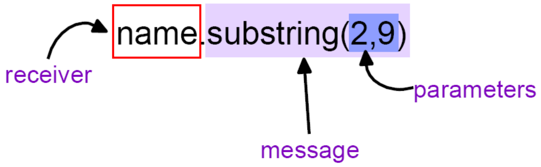
Example 3

```
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");
  }
}
```

The constructor BankAccount() is undefined

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



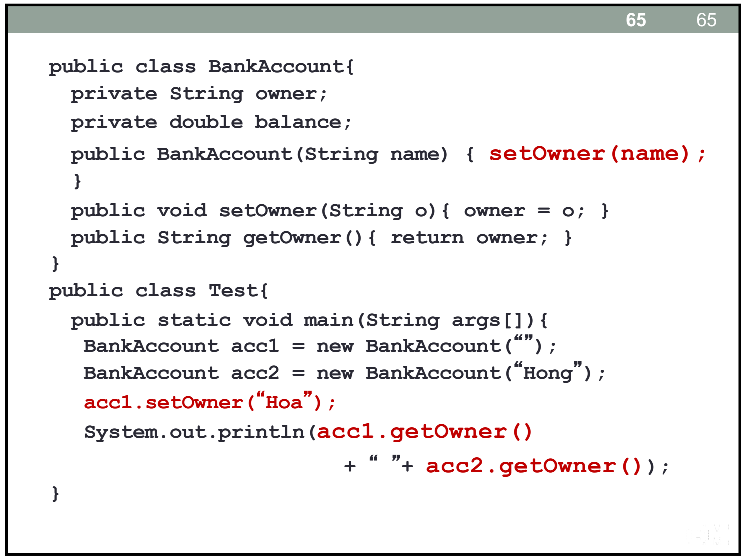
3.4. Object usage (2)

- 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.

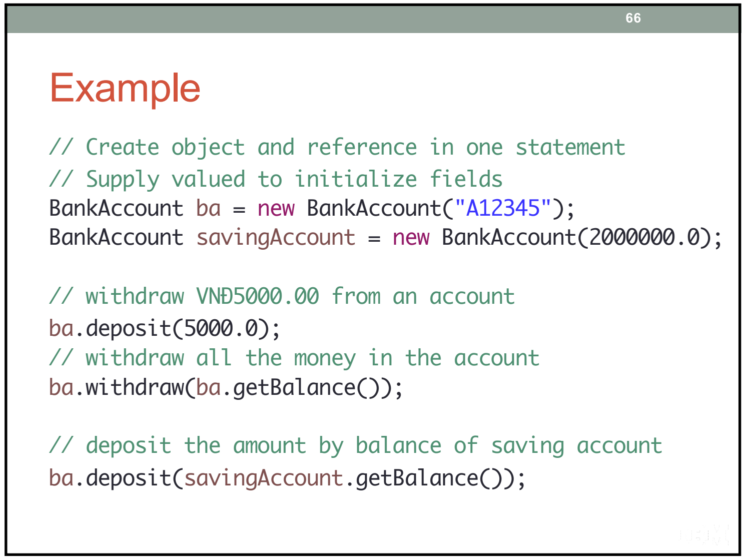
```
BankAccount account = new BankAccount();
account.setOwner("Smith");
account.credit(1000.0);
System.out.println(account.getBalance());
...
```

BankAccount method

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

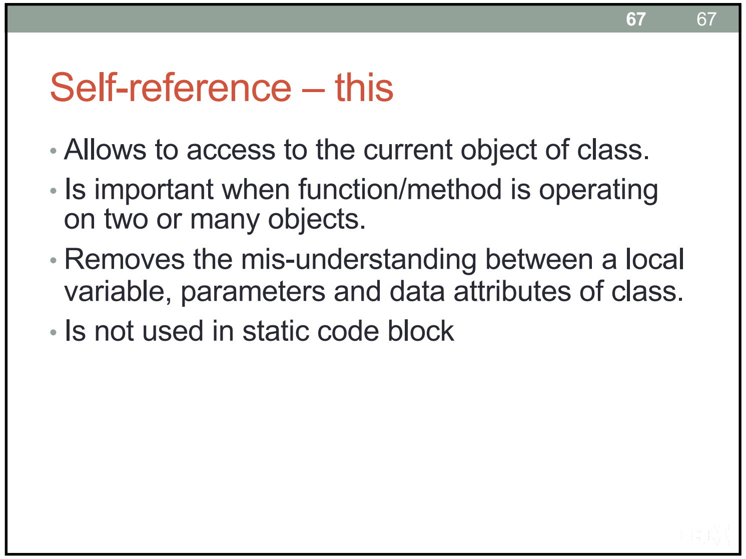


```
public class BankAccount{
  private String owner;
  private double balance;
  public BankAccount(String name) { setOwner(name);
  }
  public void setOwner(String o){ owner = o; }
  public String getOwner(){ return owner; }
}
public class Test{
  public static void main(String args[]){
   BankAccount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



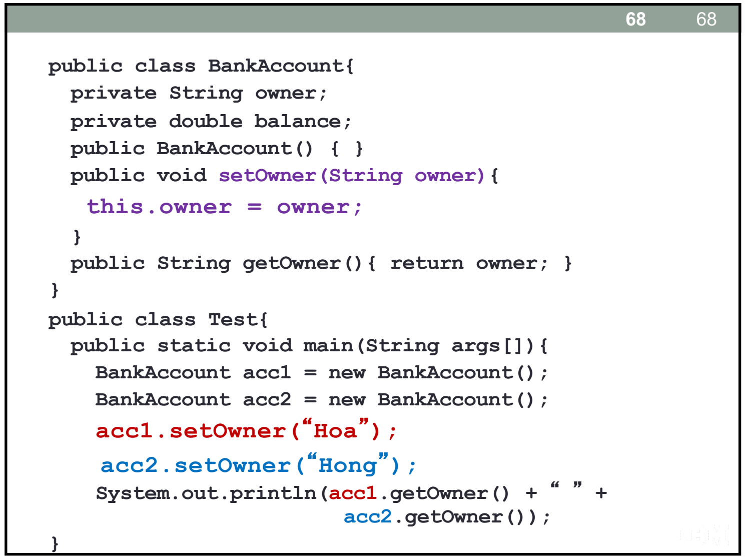
Example

```
// Create object and reference in one statement
// Supply valued to initialize fields
BankAccount ba = new BankAccount("A12345");
BankAccount savingAccount = new BankAccount(2000000.0);

// withdraw VNĐ5000.00 from an account
ba.deposit(5000.0);
// withdraw all the money in the account
ba.withdraw(ba.getBalance());

// deposit the amount by balance of saving account
ba.deposit(savingAccount.getBalance());
```



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



```
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());
}
```