OBJECT-ORIENTED LANGUAGE AND THEORY

3. ABSTRACTION & ENCAPSULATION

BÁCH KHOA

1

4 principles of object-oriented programming

- 4 principles: **abstraction, inheritance, encapsulation**, and **polymorphism**
  - Abstraction lets us selectively focus on the high-level and abstract way the low-level details.
  - Inheritance is about code reuse, not hierarchies.
  - Encapsulation keeps state private so that we can better enforce business rules, protect model invariants, and develop a single source of truth for related data and logic.
  - Polymorphism provides the ability for us to design for dynamic runtime behavior, easy extensibility, and substitutability.
- Following the principles, we can write more testable, flexible, and maintainable code.

2

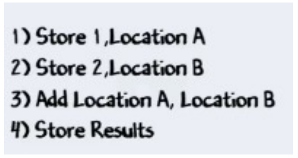
Outline

1. Abstraction
2. Class Building
3. Encapsulation and data hiding
4. Object Creation and Communication

3

1.1. 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
  - i = 1 + 2


1) Store 1,Location A
2) Store 2,Location B
3) Add Location A, Location B
4) Store Results

4

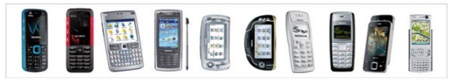
1.2. Abstraction in OOP

- 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

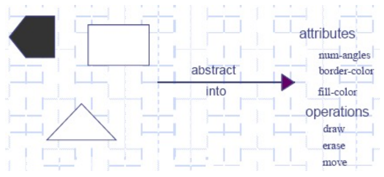
5

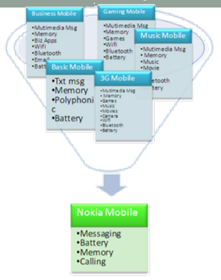
Example: Abstracting Nokia phones

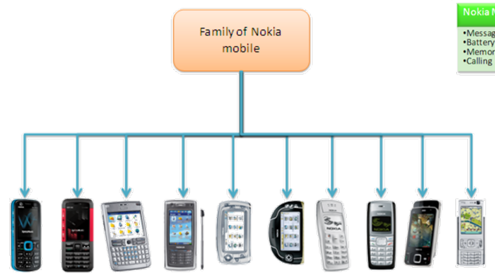


- What are the common properties of these entities? What are particular properties?
  - All are Nokia phones
  - Sliding, folding, …
  - Phones for Businessman, Music, 3G
  - QWERTY keyboard, Basic Type, No-keyboard type
  - Color, Size, …

6







7

Abstraction Example

- How your TV turns on when you press the ON button on the remote?
- How your motorbike starts when you turn your key in the ignition?
- Abstraction makes technology easier to use.
  - Most of us know that pressing the ON button on the remote will turn on a TV. That's good enough for us.
  - Imagine that we needed to know the low-level electronic details in order to turn your TV on. The learning curve would be tremendous. Very few people would watch TV if that were the case.

8

Abstraction of a washing machine

```
// Options for the wash cycle
type WashOptions = {
  dryLevel: 'low' | 'medium' | 'high'
  temperature: 'cold',
  duration: 'hour',
  ecoEnabled: false
}

// The abstraction
class WashingMachine {

  // Private instance variables
  ...

  public startCycle (options: WashOptions): void {
        // Parse the options
    // Get access to the physical layer
    // Convert options into commands
    // Lots of low-level code
    // And so on...
    ...
  }

  // More methods
  ...
}
```

9

1.2. 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

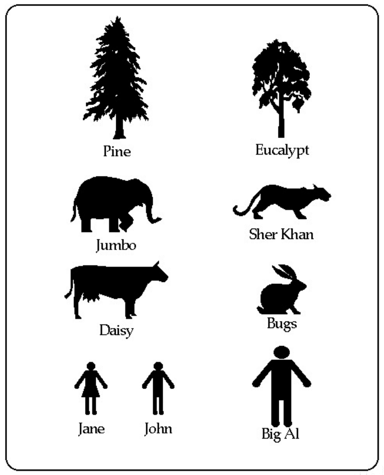
10

1.2. Abstraction (4)

- **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

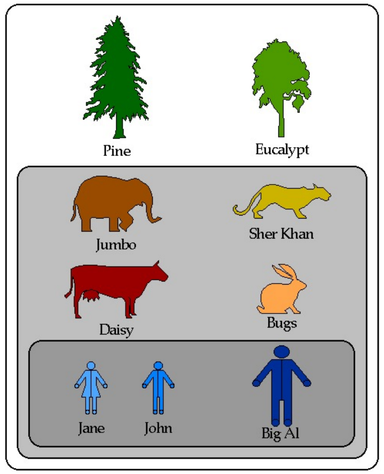
11

unclassified "things"



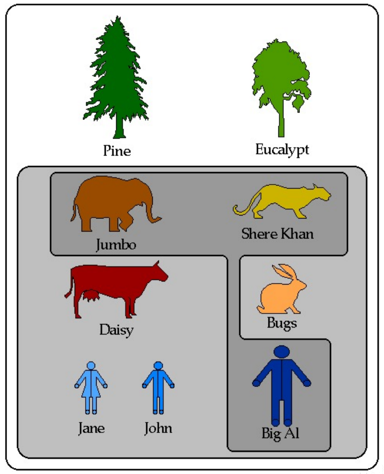
12

- organisms, mammals, humans



13

- organisms, mammals, dangerous mammals



14

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

15

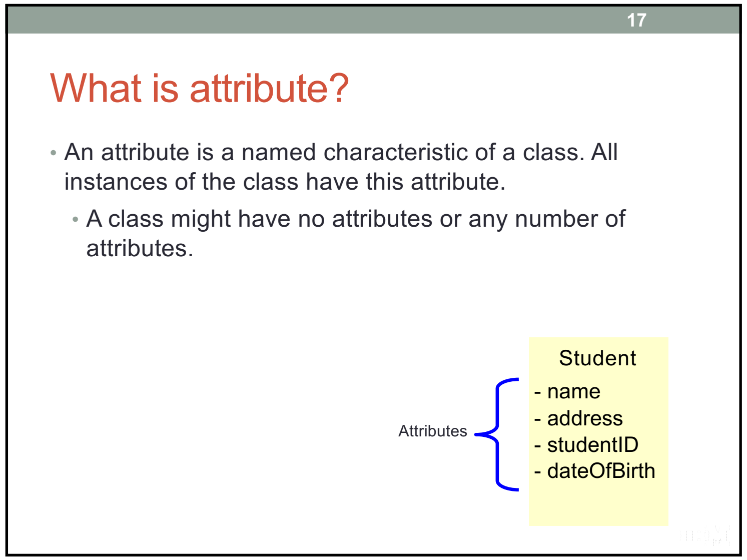
Class representation in UML

Professor

- 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()
```

16



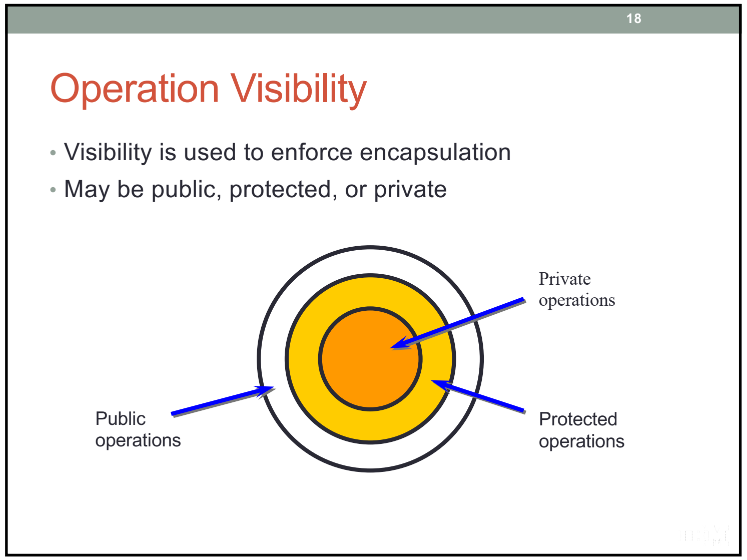
What is attribute?

- An attribute is a named characteristic of a class. All instances of the class have this attribute.
  - A class might have no attributes or any number of attributes.

Attributes

Student
- name
- address
- studentID
- dateOfBirth

17



Operation Visibility

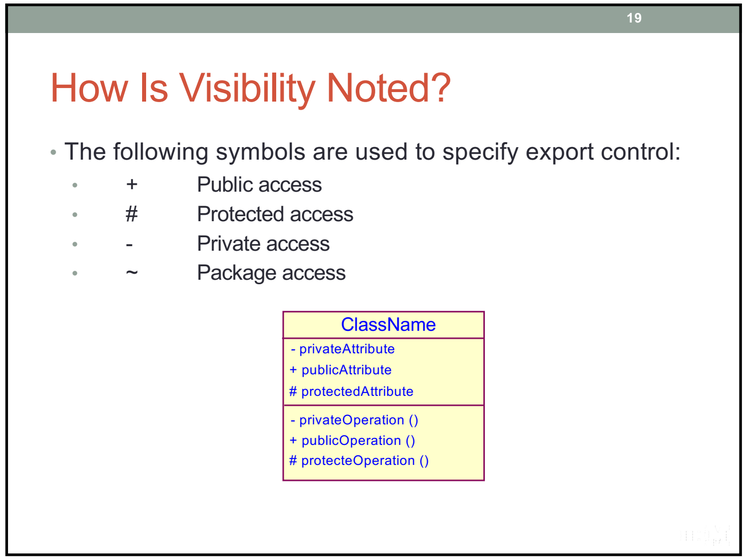
- Visibility is used to enforce encapsulation
- May be public, protected, or private

Private operations

Public operations

Protected operations

18



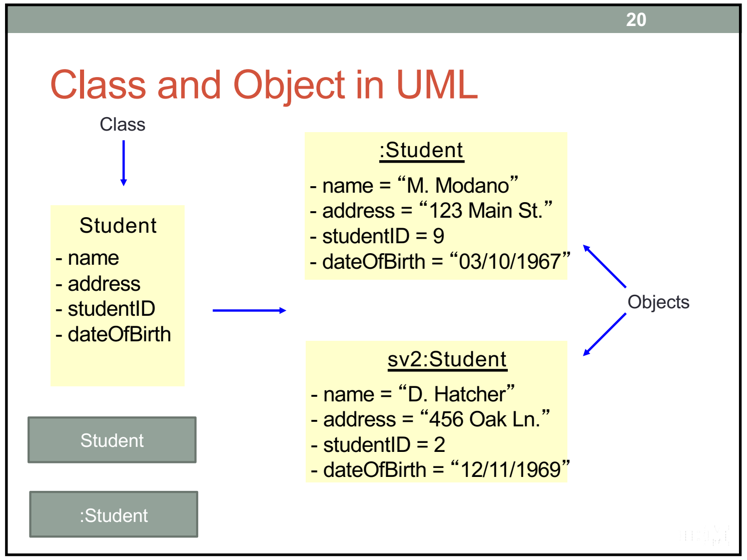
How Is Visibility Noted?

- The following symbols are used to specify export control:

+	Public access
#	Protected access
-	Private access
~	Package access

ClassName
- privateAttribute
+ publicAttribute
# protectedAttribute
- privateOperation ()
+ publicOperation ()
# protecteOperation ()

19



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”

sv2:Student
- name = “D. Hatcher”
- address = “456 Oak Ln.”
- studentID = 2
- dateOfBirth = “12/11/1969”

Objects

Student

:Student

20

Outline

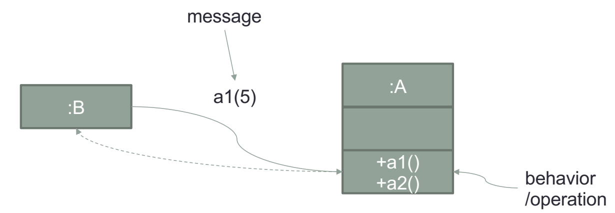
1. Abstraction
2. Class Building
3. Encapsulation and data hiding
4. Object Creation and Communication

2.2. Class Construction

BankAccount
- owner: String
- balance: double

+ debit(double): boolean
+ credit(double)

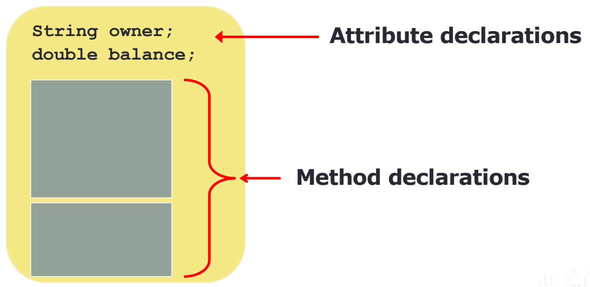
- **Class name**
  - Specify what the abstraction is capturing
  - Should be singular, short, and clear identify the concept
- **Data elements**
  - The pieces of data that an instance of the class holds
- **Operations/Messages**
  - List of messages that instances can receive
- **Methods**
  - Implementations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



```
class A {
            - attr1;
            - attr2;
            - attr3;
            - attr4;

            +a1(int i){
                        //tính tiền trong ví
method                  //tính tiền trong tài khoản…
                        //trả về tổng số tiền
            }
}
```

2.2. Class Construction (2)

- Class members
  - Attributes/Fields
  - Methods



```
String owner;
double balance;
```

Package

- 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.
- Package help us to avoid **name conflict**: different packages can have classes with the same name
- Package can **protect** class and their members from outside access
- Package in other programming language (C#, ++) is known as **namespace**

Package

- 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: use lower letters only
  - Example: `package oolt.hedspi;`
- Each source file can have only one **package declaration** command **at the top**. If we use not use this command, the file is in **default package**

a. Class declaration

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

- Declaration syntax:

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

- `access_modifier:`
  - `public`: Class can be accessed from anywhere, including outside its package.
  - `None (default)`: Class can be access from inside its package

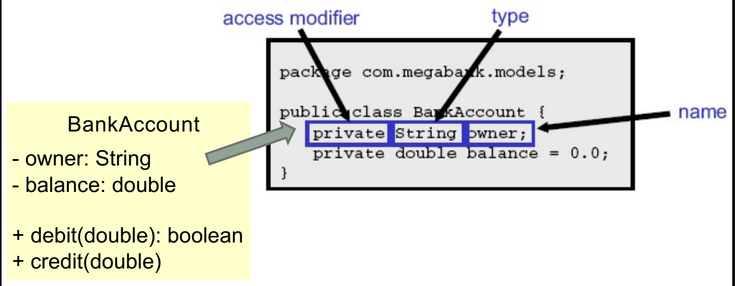
Attribute

- 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

Attribute

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

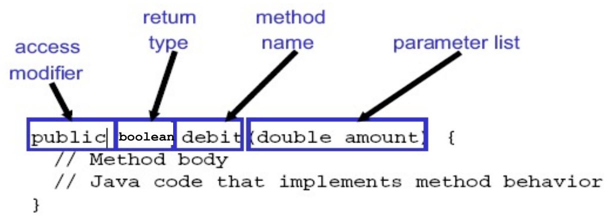


```
package com.megabank.models;

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

29

Method

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

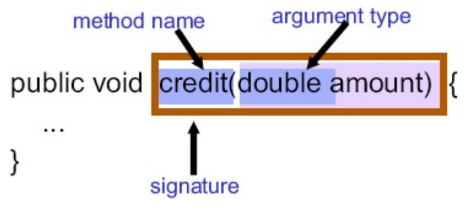


```
public boolean debit(double amount) {
   // Method body
   // Java code that implements method behavior
}
```

30

* Method signature

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



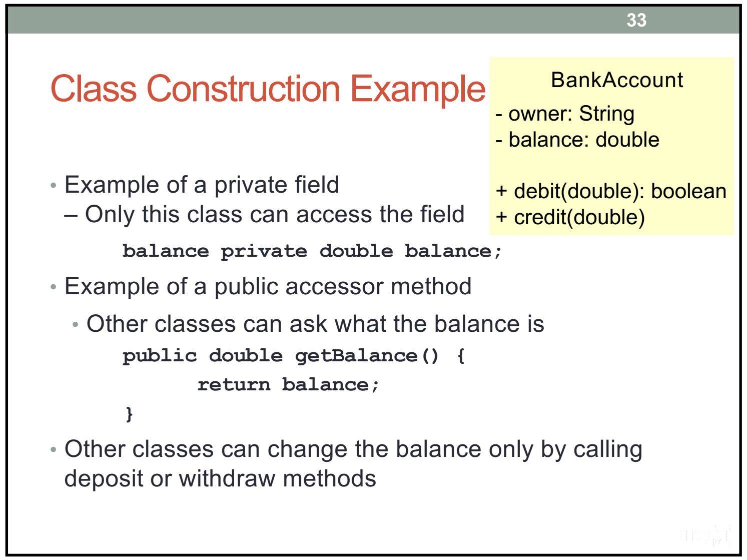
```
public void credit(double amount) {
   ...
}
```

31

* 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” commands in a method; the first one that is reached will be executed.

32



Class Construction Example

BankAccount
- owner: String
- balance: double

+ debit(double): boolean
+ credit(double)

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

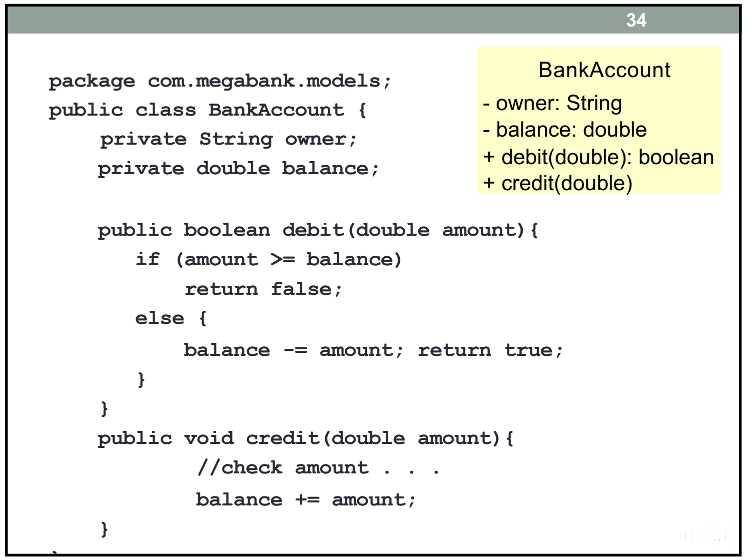
```
balance private double balance;
```

- 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

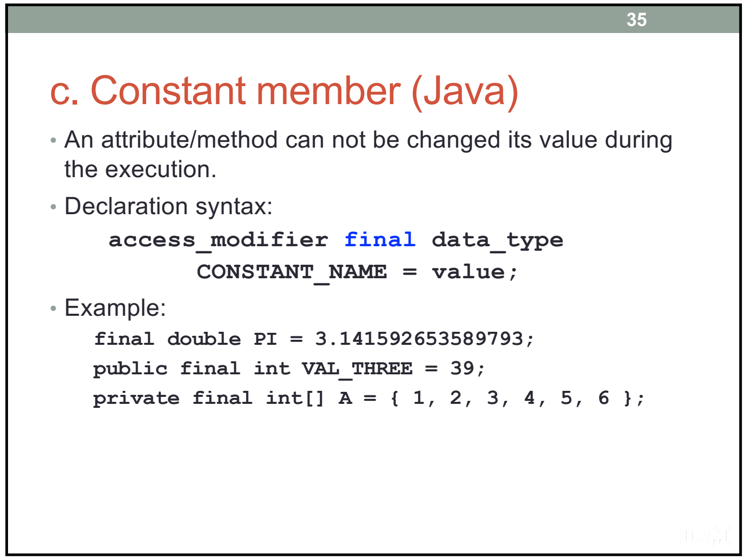
33



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

```
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){
            //check amount . . .
            balance += amount;
    }
```

34



c. Constant member (Java)

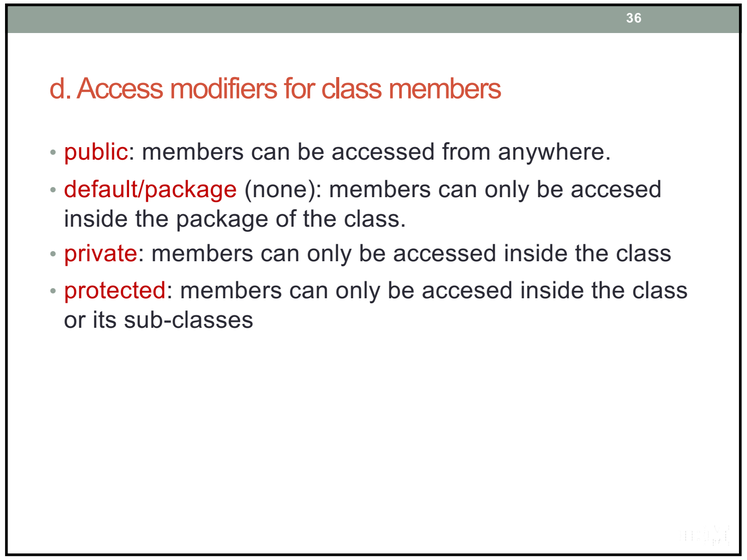
- An attribute/method can not be changed its value during the execution.
- Declaration syntax:

```
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 };
```

35



d. Access modifiers for class members

- public: members can be accessed from anywhere.
- default/package (none): members can only be accesed inside the package of the class.
- private: members can only be accessed inside the class
- protected: members can only be accesed inside the class or its sub-classes

36

d. Access modifiers for class members

	public	None	private
Same class			
Same package			
Different package			

d. Access modifiers for class members

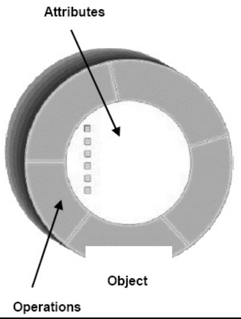
	public	None	private
Same class	Yes	Yes	Yes
Same package	Yes	Yes	No
Different package	Yes	No	No

Outline

1. Abstraction
2. Class Building
3. **Encapsulation and data hiding**
4. Object Creation and Communication

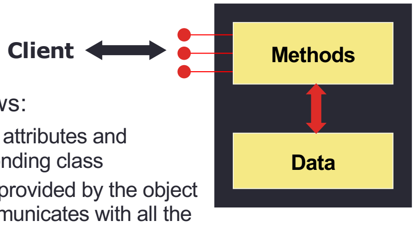
Encapsulation (2)

- Data/attributes and behaviors/methods are encapsulated in a class → Encapsulation
  - Attributes and methods are members of the class



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

Encapsulation

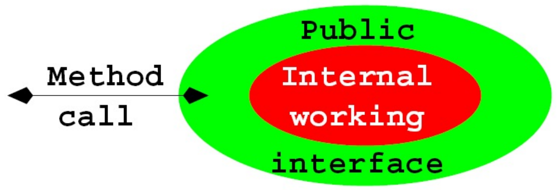


- An object has two views:
  - Internal view: Details on attributes and methods of the corresponding class
  - External view: Services provided by the object and how the object communicates with all the rest of the system

41

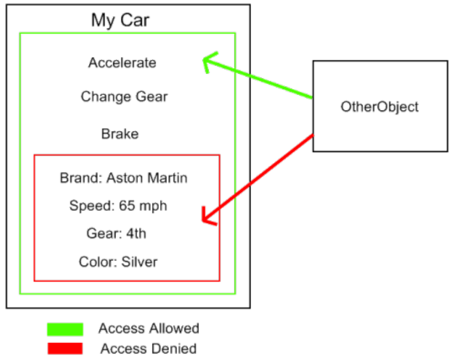
Data hiding

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



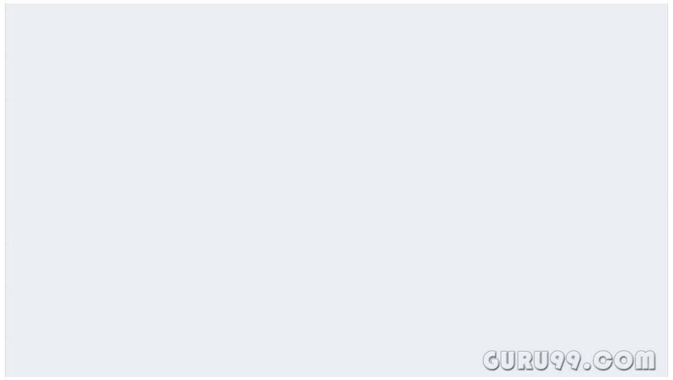
42

Example – Data hiding



43

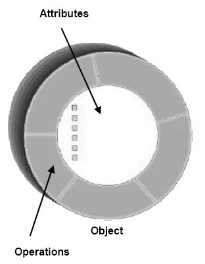
Encapsulation with Java



44

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)

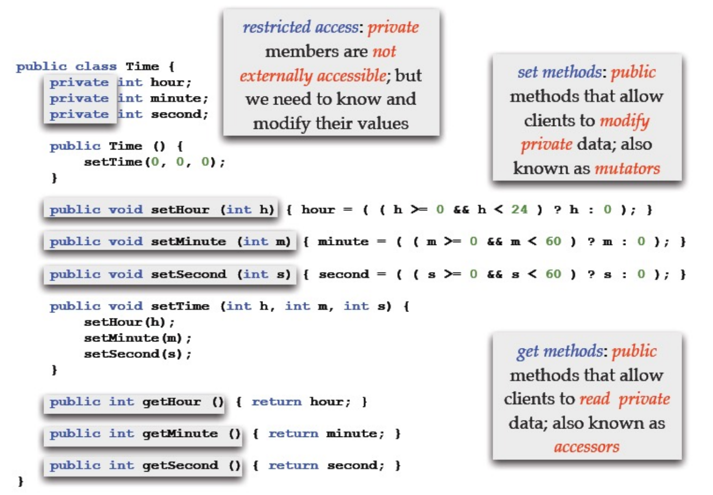
- Because data is private → Normally a class provides services to access and modify values of the data
  - **Accessor** (getter): return the current value of an attribute
  - **Mutator** (setter): modify value of an attribute
  - Usually getX and setX, where x is attribute name

```
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

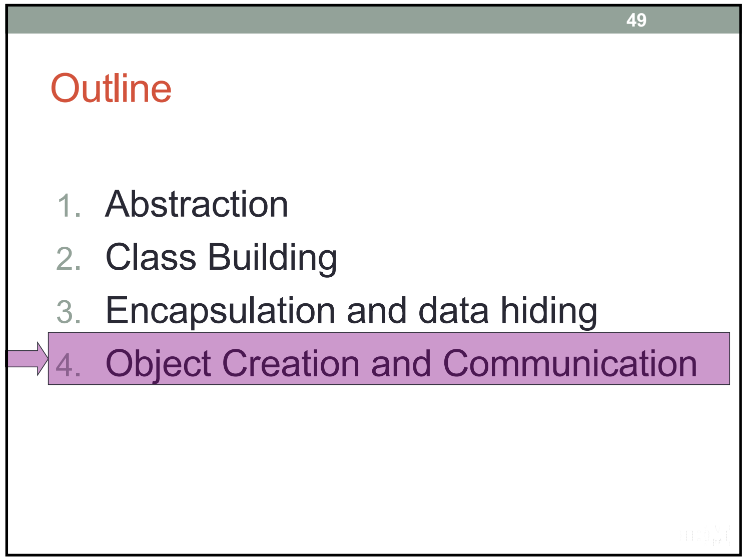


```
public class Time {
    private int hour;
    private int minute;
    private int second;

    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; }
}
```

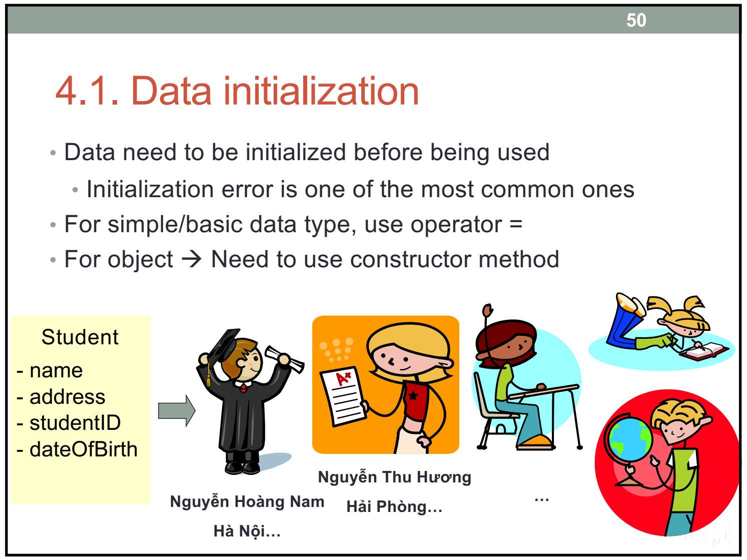
restricted access: *private* members are *not externally accessible*; but we need to know and modify their values

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*



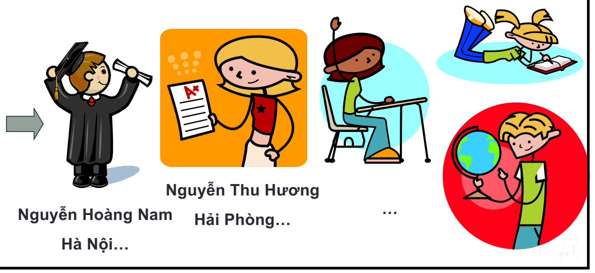
Outline

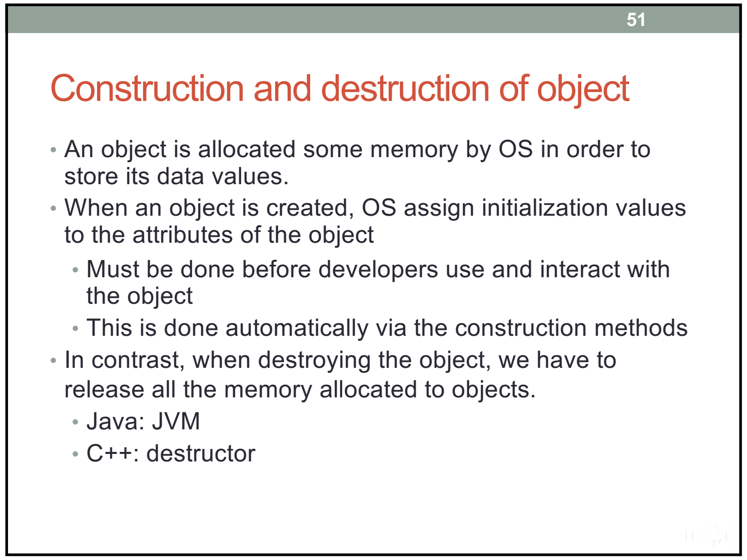


4.1. Data initialization

- 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

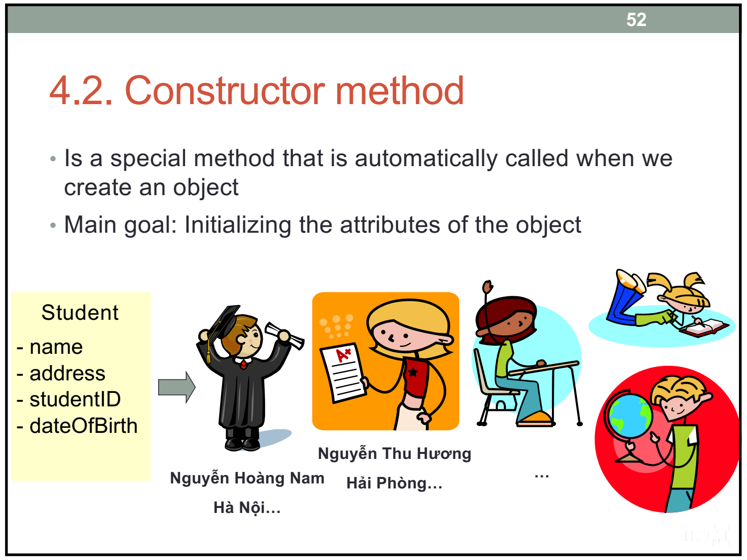
Student
- name
- address
- studentID
- dateOfBirth





Construction and destruction of object

- An object is allocated some memory by OS in order to store its data values.
- When an object is created, OS assign initialization values to the attributes of the object
  - Must be done before developers use and interact with the object
  - This is done automatically via the construction methods
- In contrast, when destroying the object, we have to release all the memory allocated to objects.
  - Java: JVM
  - C++: destructor



4.2. Constructor method

- Is a special method that is automatically called when we create an object
- Main goal: Initializing the attributes of the object

Student
- name
- address
- studentID
- dateOfBirth

4.2. Constructor method

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

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

53

4.2. Constructor method

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

54

Types of constructor methods

- Constructor methods that have no parameters: default constructor
- Constuctor methods that have parameters

55

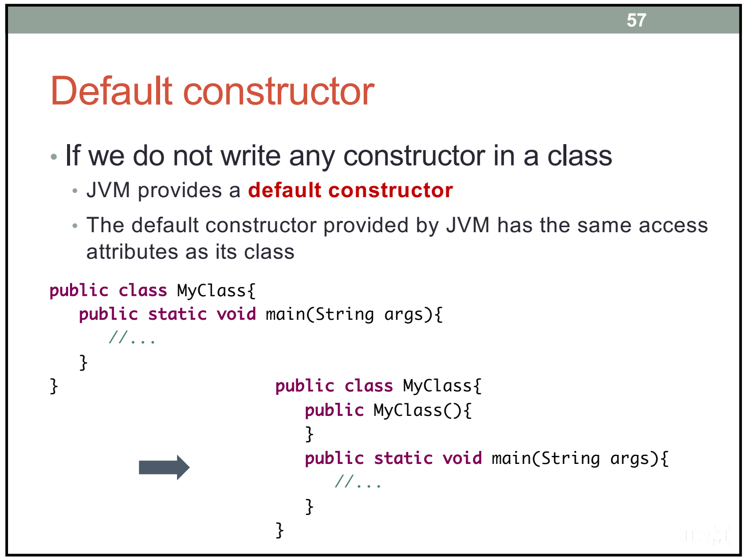
Default constructor

- Is a constructor **without parameters**

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

- A class should have a default constructor

56



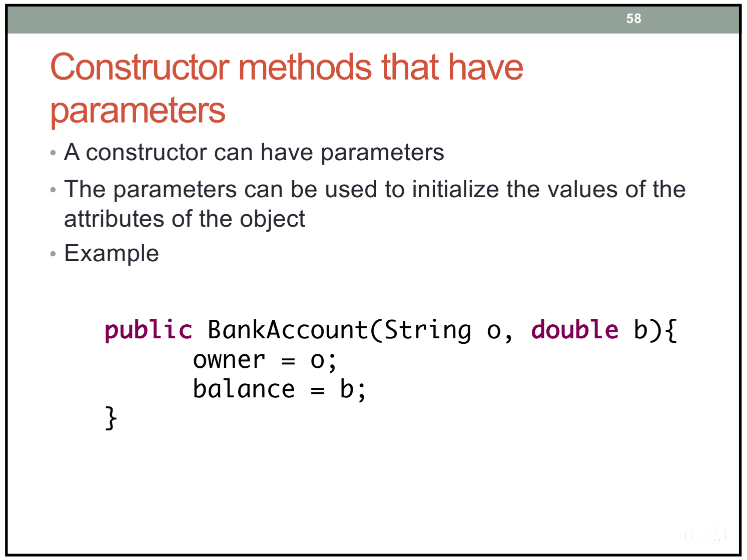
Default constructor

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

```
public class MyClass{
   public static void main(String args){
      //...
   }
}
```

```
public class MyClass{
   public MyClass(){
   }
   public static void main(String args){
      //...
   }
}
```

57

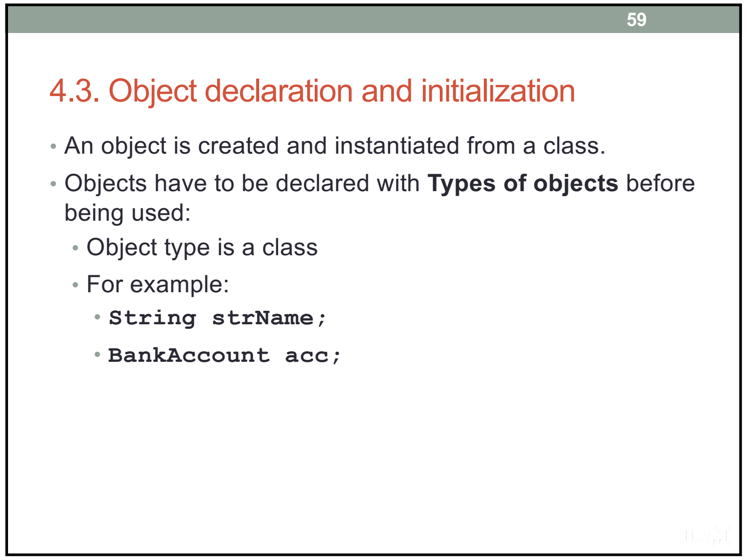


Constructor methods that have parameters

- A constructor can have parameters
- The parameters can be used to initialize the values of the attributes of the object
- Example

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

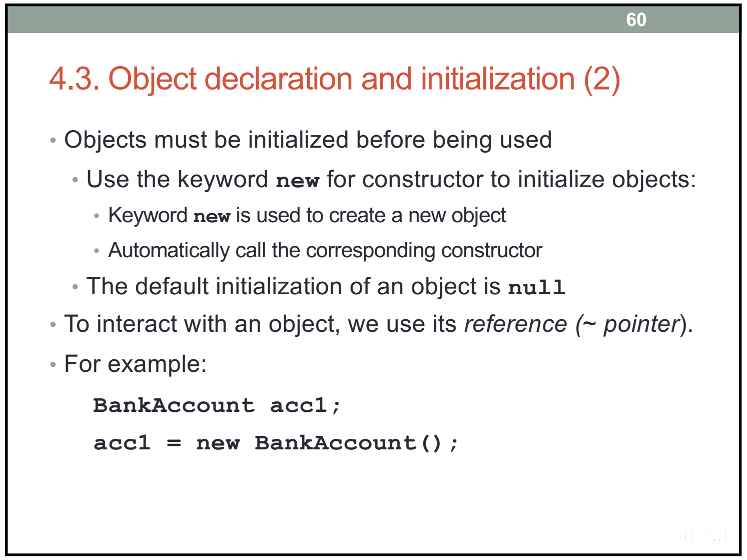
58



4.3. 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 a class
  - For example:
    - `String strName;`
    - `BankAccount acc;`

59



4.3. Object declaration and initialization (2)

- Objects must be initialized before being used
  - 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`**
- To interact with an object, we use its *reference* (~ *pointer*).
- For example:

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

60

4.3. 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();
```

4.3. 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");
```

4.3. Object declaration and initialization (5)

- Array of objects is declared similarly to an 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.

65

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

66

Objects in C++ and Java

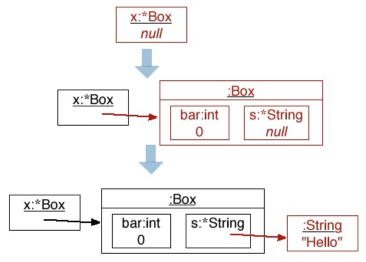
- C++: objects in a class are created at the declaration:
  - Point p1;
- Java: Declaration of an object creates only a reference that will refer to the real object when **new** operation is used:
  - Box x;
  - x = new Box();
  - Objects are dynamically allocated in heap memory

67

Object in Java

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

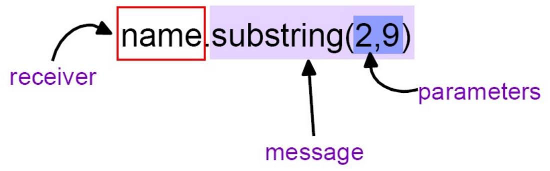
```
Box x;

x = new Box ();

x.s = "Hello";
```



68

4.4. 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



69

4.4. 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);
}
```

70

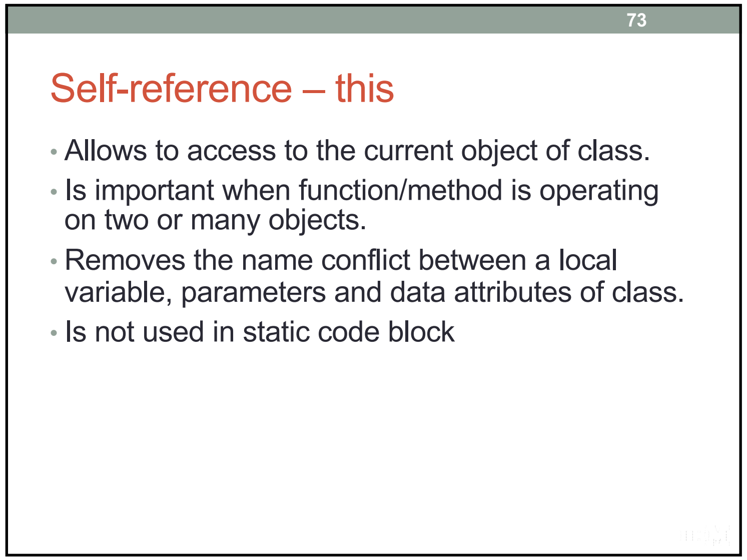
```
public class BankAccount{
  private String owner;
  private double balance;
  public BankAccount(String name){ setOwner(name);}
  public void setOwner(String o){ owner = o; }
  public String getOwner(){ return owner; }
}
public class Test{
  public static void main(String args[]){
   BankAccount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

71

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());
```

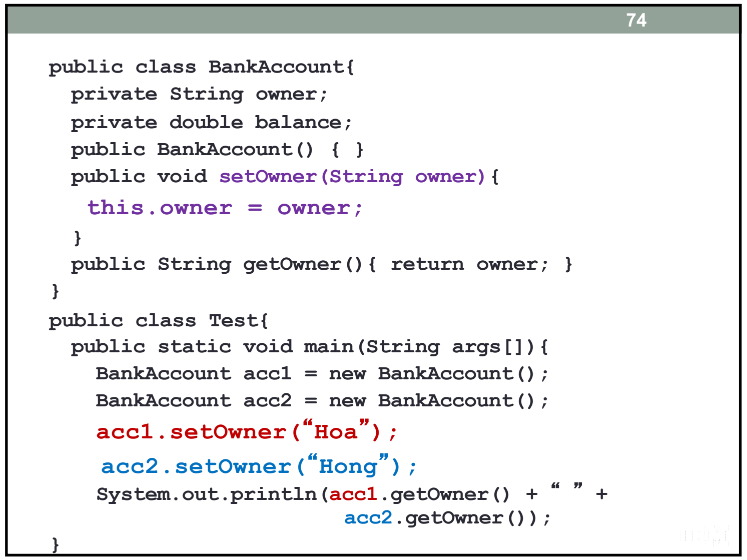
72



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 name conflict between a local variable, parameters and data attributes of class.
- Is not used in static code block

73



```
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());
}
```

74