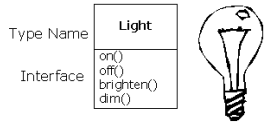
2.5 An object has an interface

- But how do you get an object to do useful work for you?
- An object can be used when it can respond to certain "requests" from outside.
- The requests you can make of an object are defined by its interface

C++

```
Light
lt;
lt.on();
```



Java

```
Light lt = new Light();
lt.on();
```


Examples of class and object in some OOP languages

Class declaration: *each class* is, by default, an *extension of* **`Object`** (can be omitted)

Class fields: **`private`** means they *can not be accessed* from *outside* the class

Class constructor: *initialises* the various *fields*

Class method: *retrieves* and/or *modifies* the *state* of the class

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

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

    public void setTime (int h, int m, int s) {
        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
        minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
        second = ( ( s >= 0 && s < 60 ) ? s : 0 );
    }
}
```

Java: Program and object

```
public class Test {

    public static void main (String args[]) {
        Time time = new Time();

        time.hour = 7;
        time.minute = 15;
        time.second = 30;
    }
}
```

```
Test.java:6: hour has private access in Time
        time.hour = 7;
             ^
Time.java:7: minute has private access in Time
        time.minute = 15;
             ^
Time.java:8: second has private access in Time
        time.second = 30;
             ^
3 errors
```


Class Time in C++

15

Class definition bắt đầu bằng từ khoá **class**.

Class body bắt đầu bằng ngoặc mở.

Function prototype cho các **public** member function.

Class Time definition (1 of 1)

Constructor: thành viên trùng tên với tên class, **Time**, và không có giá trị trả về.

Nhãn quyền truy nhập

private data member chỉ có thể được truy nhập từ các member function.

```
1  class Time {
2
3  public:
4     Time();                        // constructor
5     void setTime( int, int, int ); // set hour, minute, second
6     void printUniversal();         // print universal-time format
7     void printStandard();          // print standard-time format
8
9  private:
10    int hour;     // 0 - 23 (24-hour clock format)
11    int minute;   // 0 - 59
12    int second;   // 0 - 59
13
14 }; // end class Time
```

Program and object: C++

```
1  // Fig. 6.7: fig06_07.cpp
2  // Program to test class Time.
3  // NOTE: This file must be compiled with time1.cpp.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  // include definition of class Time from time1.h
10 #include "time1.h"
11
12 int main()
13 {
14    Time t;  // instantiate object t of class Time
15
16    // output Time object t's initial values
17    cout << "The initial universal time is ";
18    t.printUniversal();   // 00:00:00
19    cout << "\nThe initial standard time is ";
20    t.printStandard();    // 12:00:00 AM
21
22    t.setTime( 13, 27, 6 );   // change time
23
```

fig06_07.cpp
(1 of 2)

Include **time1.h** để đảm bảo tạo đúng và để tính kích thước đối tượng thuộc lớp **Time**.

Program and object: C++

```
24    // output Time object t's new values
25    cout << "\n\nUniversal time after setTime is ";
26    t.printUniversal();   // 13:27:06
27    cout << "\nStandard time after setTime is ";
28    t.printStandard();    // 1:27:06 PM
29
30    t.setTime( 99, 99, 99 );  // attempt invalid settings
31
32    // output t's values after specifying invalid values
33    cout << "\n\nAfter attempting invalid settings:"
34         << "\nUniversal time: ";
35    t.printUniversal();   // 00:00:00
36    cout << "\nStandard time: ";
37    t.printStandard();    // 12:00:00 AM
38    cout << endl;
39
40    return 0;
41
42 } // end main
```

fig06_07.cpp
(2 of 2)

fig06_07.cpp
output (1 of 1)

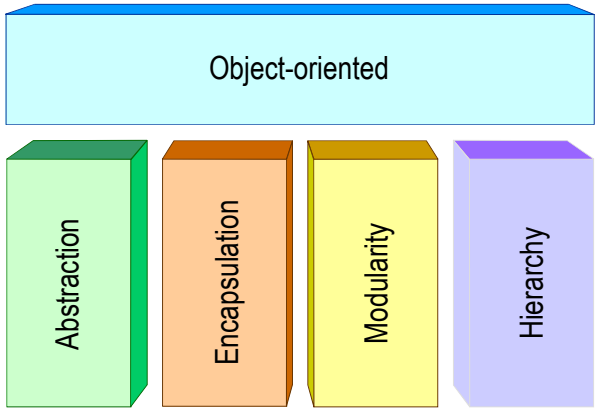
```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM
```

Outline

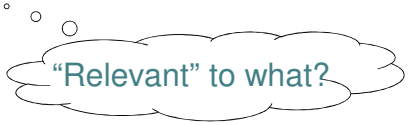
1. Object-oriented Engineering
2. Object and Class
3. Principles of OO
4. Object-oriented analysis and design
5. Java/C++ programming languages
6. Examples and Exercises

3. Basic principles of OO



2.1 Abstraction

- ***"Abstraction is selective ignorance"***
- Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones…



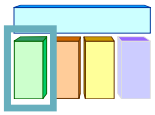
… relevant to the given project (with an eye to future reuse in similar projects).

Abstraction

- Decide what is important and what is not
- Focus and depend on what is important
- Ignore and do not depend on what is unimportant
- Use encapsulation to enforce an abstraction

Abstraction

- Is a process to remove detailed information and keep only common information.
- Focus on essential characteristics of entities, on characteristics to distinguish them from other entities.
- Depend on perspective (context)
  - When you drive a car, thinking about how the engine works is a distraction
  - When you repair a car, thinking about how the engine works is essential

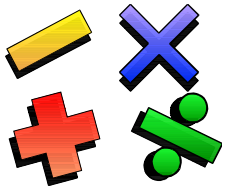


Example: Abstraction

Student

Teacher

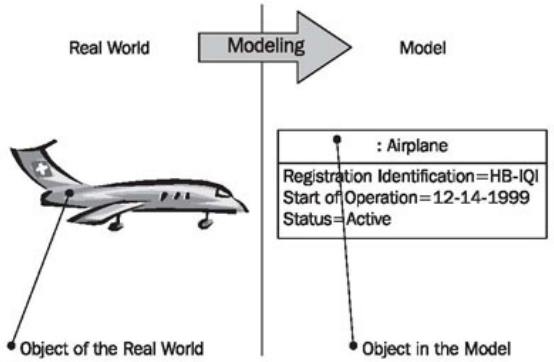
Course starts at 9:00 every morning of Tuesday, Thursday and Saturday



Course (Algebra for example)

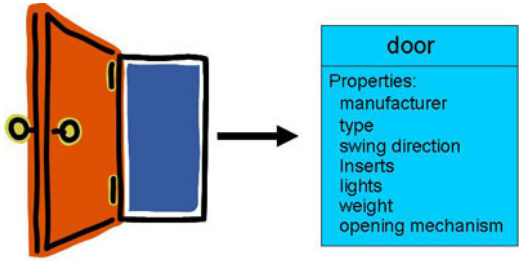
61

Abstraction



62

Example: Door

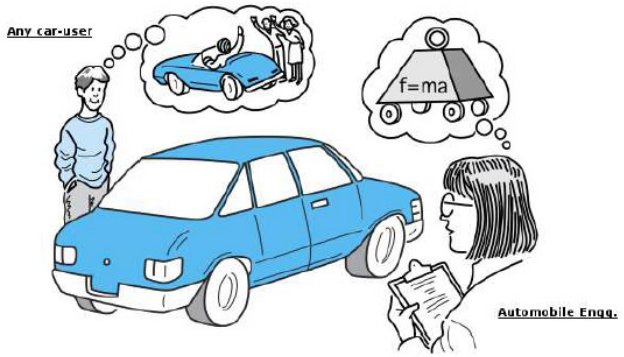


Abstraction

Don't need to know this

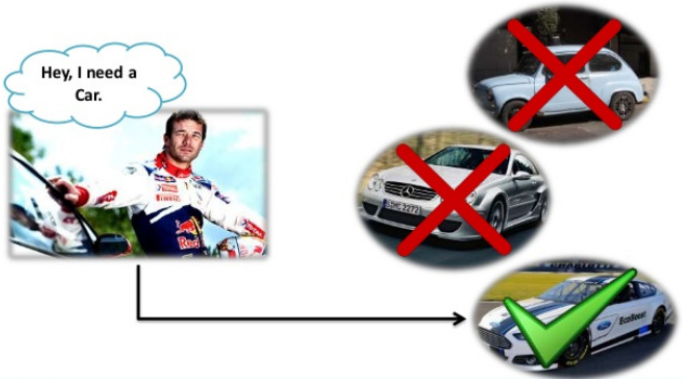
Can focus on this!!

Depend on perspectives



65

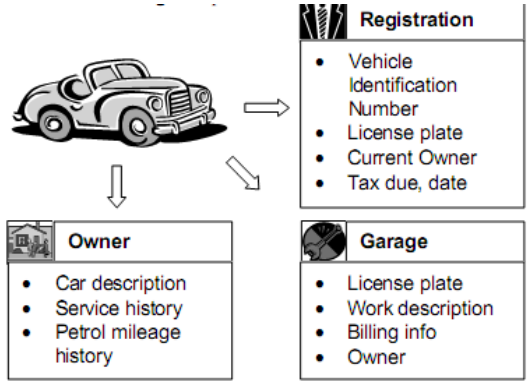
You are the salesman, please help the customer..



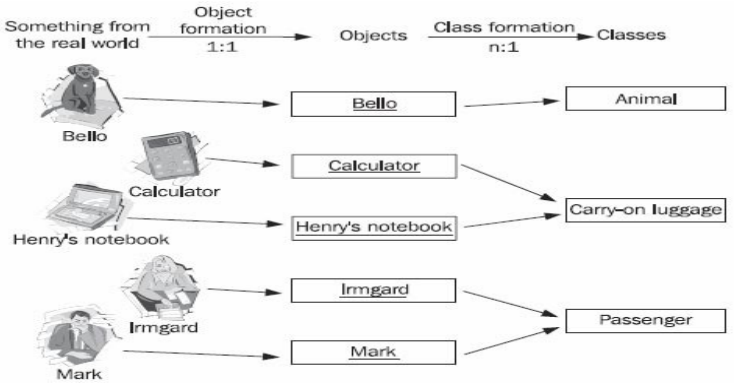
66

You are the salesman, please help the customer..

Abstraction – different perspectives



From object to class



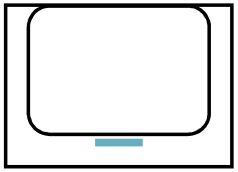
Questions at the class

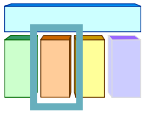
- An only system for making friends
  - User's perspective
  - Admin's perspective

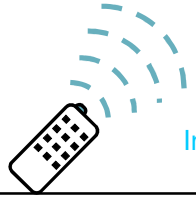
2.2. Encapsulation

- Process of combining data (properties) and functions acting on that data (methods) into a single unit
- Process of hiding object's implementation from another object, while presenting only the interface
- Programmer can specify certain methods or state variables remain hidden

- Provide an interface to outside
- Users does not depend on some implementation changes inside





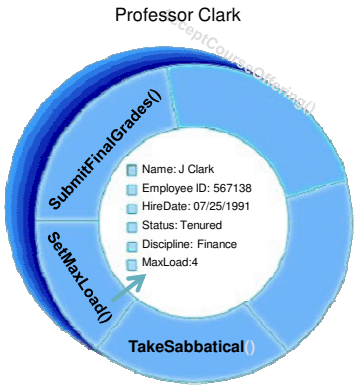


Increasing flexibility

Encapsulation illustration

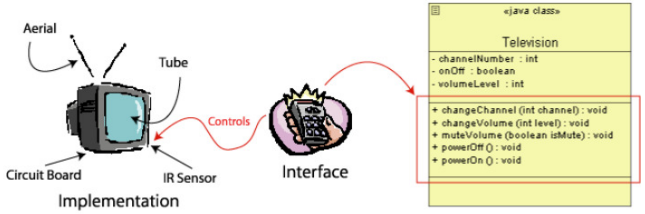
- Interface messages (methods) of the object
- Professor Clark is required to teach 4 classes next month

SetMaxLoad(4)



Encapsulation

- Two aspects of encapsulation
  - Data Hiding or information hiding
  - Hidden implementation



Data hiding

- Data are accessible within the object, but not outside it
- Restricting access to some methods (« public methods »)
  - Accessors: is a method that is used to ask an object about itself. In OOP, these are usually in the form of properties, which have, under normal conditions.
  - Mutators: are public methods that are used to modify the state of an object, while hiding the implementation of exactly how the data gets modified.

Benefits of data hiding

- Data protection
  - Avoid misuse of data
  - Any change can be effected indirectly only via set of public method
- Easy to modify data
  - Dictionary – may use Linked List, BST or Hash Table..
  - The user (who could be another programmer) need only understand the interface, not how.

Hidden implementation

- Class creator and client programmers.
- Hidden implementation allows:
  - Specifying that users can only access and use what are specified for them. A part of class is hidden and is not accessible by users.
  - Class designers to be able to modify or re-define some methods of class and make sure that there is no problem for program using the class.

Review Quiz

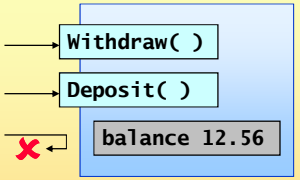
- What are advantages of the encapsulation in summary?

Encapsulation ensures that structural changes remain local

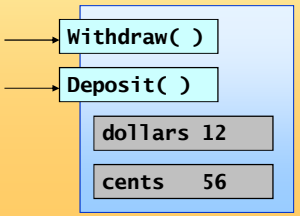
- Changes in the code create software maintenance problems
- Usually, the structure of a class (as defined by its fields) changes more often than the class's constructors and methods
- Encapsulation ensures that when fields change, no changes are needed in other classes (a principle known as "locality")

Encapsulation

- Allows to control
  - Object usage is controlled by public methods



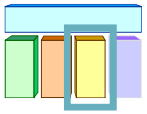
- Support modification
  - Object usage is not affected if private data is modified

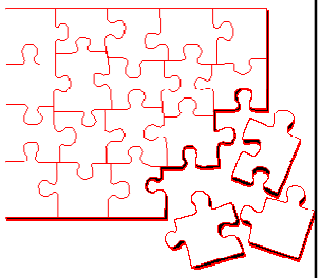


Abstraction and Encapsulation

2.3. Modularity

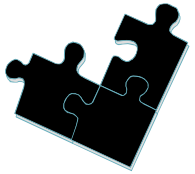
- Splitting a complex system into smaller components that can be managed.
- Allows users to understand the system.





Example: Modularity

- For example, splitting a complex system into smaller modules.



Management system for book supermarket

Book export management system

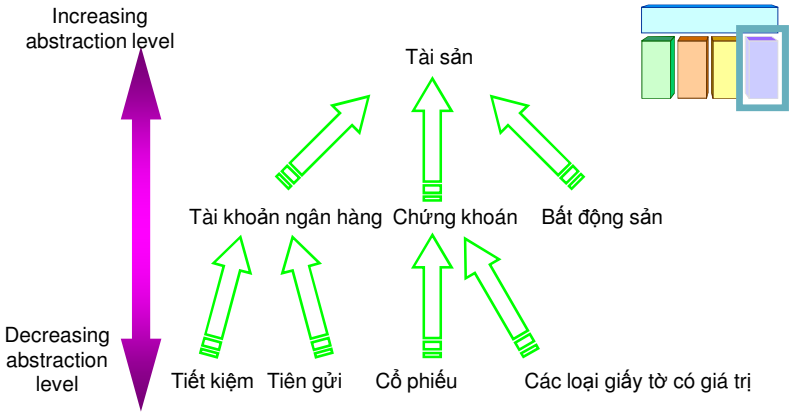
Accountin g system

Staff manageme nt system

2.4. Hierarchy

- **Hierarchy** can be defined as:
  - the ranking or order of the abstraction level in a tree structure. Types: aggregation hierarchy, class hierarchies, hierarchical scope, the inheritance hierarchy, hierarchical components, hierarchical level specialization, hierarchy types. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995.)

2.4. Hierarchy



Elements at the same level in the hierarchy diagram have the same level of abstraction

Outline

1. Object-oriented Engineering
2. Object and Class
3. Principles of OO
4. Object-oriented analysis and design
5. Java/C++ programming languages
6. Examples and Exercises

Analysis and Design methodologies

- Methodology: a formal set of processes and heuristics to analyze, design and build a software system
- the goal of an OOAD methodology is to discover:
  - What are the objects? (How do you partition your system?)
  - What are their interfaces? (What messages are sent and handled?)

Object oriented analysis and design

- Six phases
  - Phase 0: Make a plan
  - Phase 1: What are we making
  - Phase 2: How we will build it
  - Phase 3: Building the core
  - Phase 4: Iterate the use cases
  - Phase 5: Evolution

87

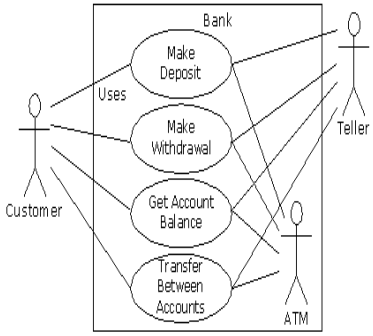
What are we making

- In the *procedural design:* Requirements analysis and system specification
  - RA: "Make a list of the guidelines we will use to know when the job is done and the customer is satisfied."
  - SP: "Here's a description of *what* the program will do (not *how*) to satisfy the requirements."
- Use cases

88

Use cases

- identify key features in the system.
- answers to questions like
  - Who will use this system?
  - What can those actors do with the system?
  - How does this actor do that with this system?
  - How else might this work if someone else were doing this, or if the same actor ha a different objective?
  - What problems might happen while doing this with the system?



89

Use cases

- A use case does not need to be terribly complex, even if the underlying system is complex.
- *It is only intended to show the system as it appears to the user.*

- You don't need to find the *complete and perfect* set of use cases for you system at the very start of the analysis stage. Many things will reveal themselves in time

90

How we will build it (design of objects)

- Determining classes
- *Class-Responsibility-Collaboration* (CRC) card.
  - name of the class
  - responsibilities of the class: what it should do, what it should response
  - collaborations of the classes: what other classes does it interact with
- Benefit?

Class Name	
Superclasses	
Subclasses	
Responsibilities	Collaborators

CRC card

BankAccount

Super Classes :

Sub Classes : SavingAccount, MarginAccount

Description :
Store the transcation record, customer data, balance, etc.

Attributes :

Name	Description
accountNumber	A unique value to identify the accounts

Responsibilities :

Name	Collaborator
Keep the latest value of the balance	Bank controller, Transcation records

BankController

Super Classes :

Sub Classes : AccountController, TranscationController, ATMController

Description :
Control the interactions between the customer and the bank system.

Attributes :

Name	Description
status	identify the status of the controller

Responsibilities :

Name	Collaborator
withDraw	Withdraw money from the bank accc

SavingAccount

Super Classes : BankAccount

Sub Classes :

Description :
Store the cash information of the customer record.

Attributes :

Name	Description
cashBalance	latest value of the cash balance

Responsibilities :

Name	Collaborator
getBalance	TranscationController, AccountControl

ATMController

Super Classes : BankController

Sub Classes :

Description :
Control the interactions between customer and the ATM terminals.

Attributes :

Name	Description
machineType	Identify the type of the ATM terminal

Responsibilities :

Name	Collaborator
checkingPassword	

Five stages of object design

- **1. Object discovery.**
  - look for external factors and boundaries, duplication of elements in the system, and the smallest conceptual units.
- **2. Object assembly.**
  - Discover the need for new members that didn't appear during discovery. The internal needs of the object may require other classes to support it.
- **3. System construction.**
  - The need for communication and interconnection with other objects in the system may change the needs of your classes or require new classes.
  - For example, you may discover the need for facilitator or helper classes

Five stages of object design

- **4. System extension.**
  - As you add new features to a system you may discover that your previous design doesn't support easy system extension. ➔ restructure parts of the system, possibly adding new classes or class hierarchies.
- **5. Object reuse.**
  - As you change a class to adapt to more new programs, the general principles of the class will become clearer.
  - Don't expect most objects from a system design to be reusable.

Guidelines for object development

1. Let a specific problem generate a class, then let the class grow and mature during the solution of other problems.
2. Discovering the classes you need (and their interfaces) is the majority of the system design. If you already had those classes, this would be an easy project.
3. Don't force yourself to know everything at the beginning; learn as you go.
4. Always keep it simple. Little clean objects with obvious utility are better than big complicated interfaces. Use an Occam's Razor approach: Consider the choices and select the one that is simplest, because simple classes are almost always best.

Outline

1. Object-oriented Engineering
2. Object and Class
3. Principles of OO
4. Object-oriented analysis and design
5. Java/C++ programming languages
6. Examples and Exercises

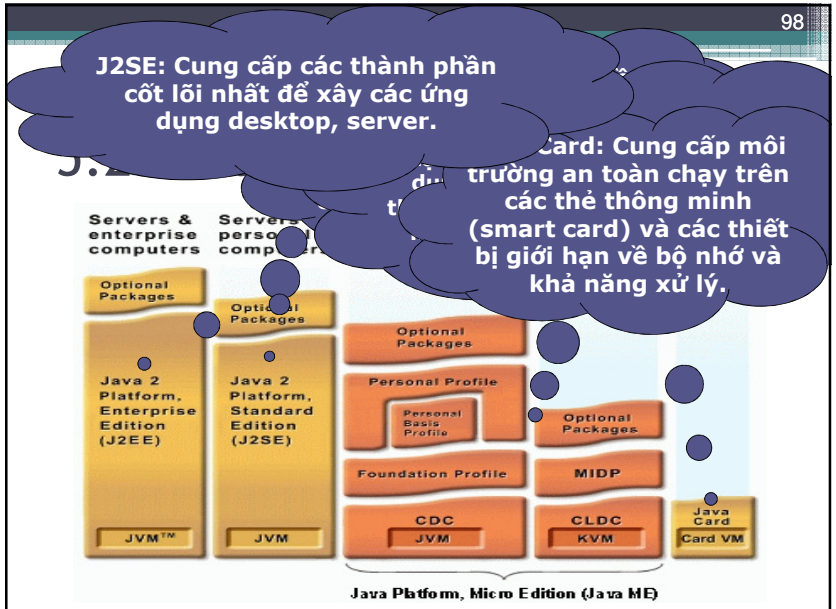
5.1 C++ language

- Extended version of C with many improvements especially OOP features.
- All existing programs of C will run in C ++
- C ++ has better features than the C language:
  - Reference variables allows managing the address of the variable or retrieve the value of the variable at the corresponding address.
  - Function name management has been extended through the function overloading mechanism.

5.2.1 Java history

- Java – language introduced by Sun Microsystems.
- Originally created for consumer electronics (TV, VCR, Freeze, Washing Machine, Mobile Phone).
  - Sun planned to use C++ for its home appliance software
  - Their new language was originally named Oak, soon changed to Java.
    - Platform independent – WORA
  - 1995: shifted the target market to Web

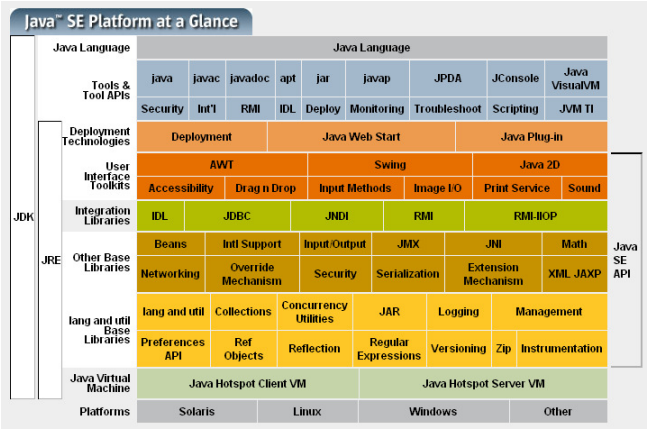
Green Team and James Gosling (the leader)



J2SE (Java 2 Platform Standard Edition)

- http://java.sun.com/j2se
- Java 2 Runtime Environment, Standard Edition (J2RE):
  - Môi trường thực thi hay JRE cung cấp các Java API, máy ảo Java (JVM) và các thành phần cần thiết khác để chạy các applet và các ứng dụng viết bằng Java.
- Java 2 Software Development Kit, Standard Edition (J2SDK)
  - Tập mẹ của JRE, và chứa mọi thứ nằm trong JRE, bổ sung thêm các công cụ như là trình biên dịch và các trình gỡ lỗi cần để phát triển applet và các ứng dụng.

J2SE (Java 2 Platform Standard Edition)



101

J2EE (Java 2 Platform Enterprise Edition)

- http://java.sun.com/j2ee
- Service-Oriented Architecture (SOA) và Web services
- Các ứng dụng Web
  - Servlet/JSP
  - JSF…
- Các ứng dụng doanh nghiệp
  - EJB
  - JavaMail…
- …

102

Lịch sử phát triển của J2SE

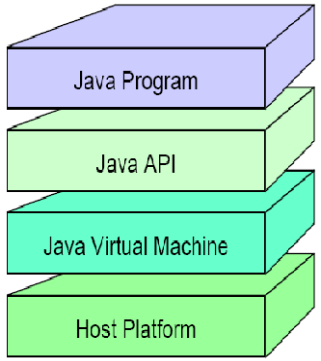
- JDK 1.1.4 (Sparkler): 12 tháng 9, 1997
- JDK 1.1.5 (Pumpkin): 3 tháng 12, 1997
- JDK 1.1.6 (Abigail): 24 tháng 4, 1998
- JDK 1.1.7 (Brutus): 28 tháng 9, 1998
- JDK 1.1.8 (Chelsea): 8 tháng 4, 1999
- J2SE 1.2 (Playground): 4 tháng 12, 1998
- J2SE 1.2.1 (none): 30 tháng 3, 1999
- J2SE 1.2.2 (Cricket): 8 tháng 7, 1999
- J2SE 1.3 (Kestrel): 8 tháng 5, 2000
- J2SE 1.3.1 (Ladybird): 17 tháng 5, 2001

Lịch sử phát triển của J2SE (2)

- J2SE 1.4.0 (Merlin) 13 tháng 2, 2002
- J2SE 1.4.1 (Hopper) 16 tháng 9, 2002
- J2SE 1.4.2 (Mantis) 26 tháng 6, 2003
- J2SE 5 (1.5.0) (Tiger) 29 tháng 9, 2004
- **Java SE 6 (Mustang)**, 11 tháng 12, 2006
  - Các bản cập nhật 2 và 3 được đưa ra vào năm 2007
  - Bản cập nhật 4 đưa ra tháng 1 năm 2008.
- **Java SE 7 (Dolphin)**, 4/2008.

5.2.3 Java platform

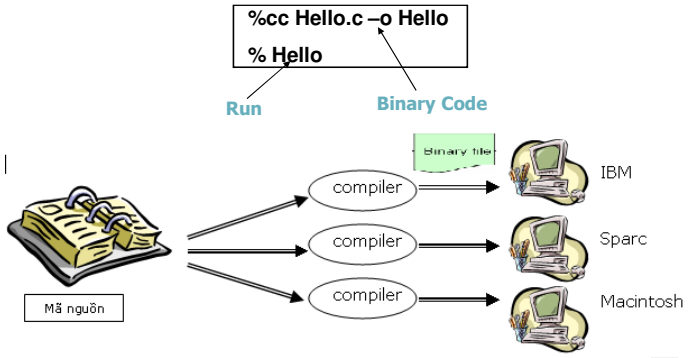
- Platform : development or deployment environment
- Java platform can run on every OS



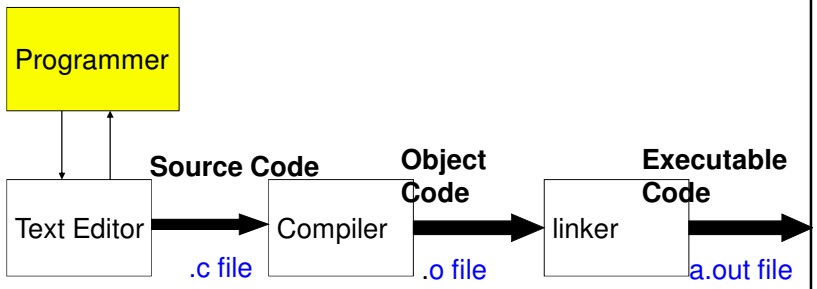
- Include
  - Java Virtual Machine (JVM).
  - Application Programming Interface (API).

5.3. Java compilation model

- a. Traditional compilation model:
  - Source code is compiled into binary code



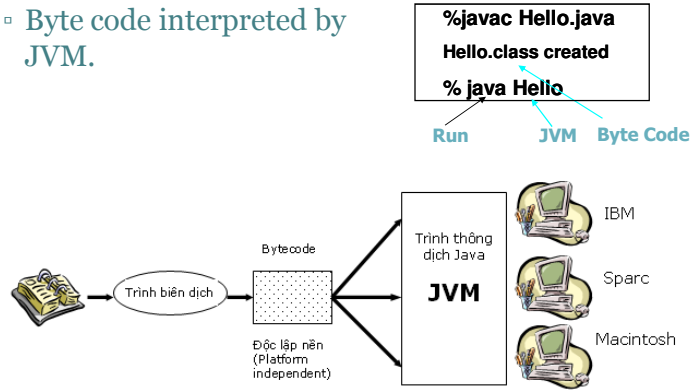
Compiled Languages



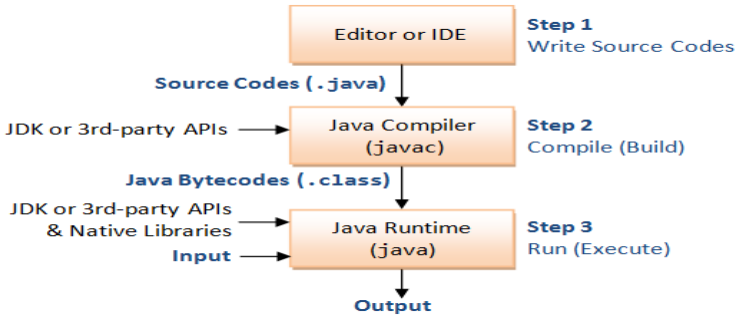
107

5.3. Java compilation model

- Source code is compiled to bytecode
- Byte code interpreted by JVM.



108



Step 1: Write the source codes (.java) using a programming text editor (such as Notepad++, Textpad, gEdit) or an IDE (such as Eclipse or NetBeans).
Step 2: Compile the source codes (.java) into Java portable bytecodes (.class) using the JDK compiler ("javac"). IDE (such as Eclipse or NetBeans) compiles the source codes automatically while they are entered.
Step 3: Run the compiled bytecodes (.class) with the input to produce the desired output, using the Java Runtime ("java").

Sun white paper defines Java as:

- Simple and Powerful
- Safe
- Object Oriented
- Robust
- Architecture Neutral and Portable
- Interpreted and High Performance
- Threaded
- Dynamic

Java Attributes

- Familiar, Simple, Small
- Compiled and Interpreted
- Platform-Independent and Portable
- Object-Oriented
- Robust and Secure
- Distributed
- Multithreaded and Interactive
- High Performance
- Dynamic and Extensible

5.5. Types of Java programs

- Stand-alone application
  - Does not need browsers to run
  - Can invoke functionalities through console or graphical user interface
  - main() method is the starting point
- Applets
  - Run in Browser - **contains no "main" method**
  - Could be viewed by appletviewer

5.5. Types of Java programs

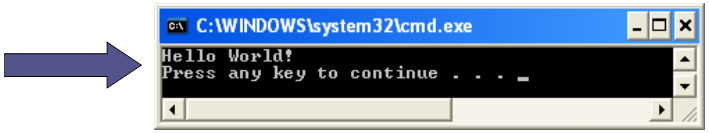
- Web application
  - Generate dynamic contents on server
  - Run in Webserver
  - Servlet: a web component that is deployed on the server to create dynamic web page.
  - JavaServer Page (JSP): HTML page embedded with Java code.

Nội dung

1. Object-oriented Engineering
2. Object and Class
3. Principles of OO
4. Object-oriented analysis and design
5. Java/C++ programming languages
6. Examples and Exercises

Example 1 - HelloWorld

```
// HelloWorld.java
// A program displaying "Hello World"
public class HelloWorld {
/* main method is called first in any Java
  application */
  public static void main(String args[]){
      System.out.println( "Hello World!" );
  } // end main method
} // end of class HelloWorld
```



Example 1 (cont..)

- Comment
  - Single line : begin with //
  - Multi line: /* ... */
- Java is case sensitive language.
- Keyword:
  - class: class declaration
  - public: access modifier
- The name of class containing main method must be the same as the source code file name .java.

Install and run a Java program

- Step 1: Install j2sdk1.5, set up environment variables (to use java in command console cmd)
- Step 2: Install TextPad/JCreator
- Step 3: Coding
- Step 4: Compile
  - cmd: javac HelloWorld.java
  - Textpad: Ctrl + 1
  - JCreator: F7 OR Build → Build Project/File
- Step 5: Run program
  - cmd: java HelloWorld.class
  - Textpad: Ctrl + 2
  - JCreator: F5 hoặc Run → Run Project/File

Environment variables

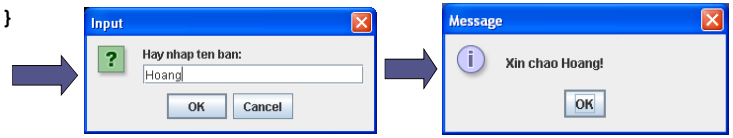
- PATH = ...;C:\Program Files\Java\jdk1.6\bin
- CLASSPATH = C:\Program Files\Java\jdk1.6\lib;C:\Program Files\Java\jdk1.6\include;

Example 2 - GUI

```
import javax.swing.JOptionPane;
public class FirstDialog{
  public static void main(String[] args){
     JOptionPane.showMessageDialog(null,
                                   "Xin chao ban!");
     System.exit(0);
  }
}
```

Example 3 – Data input and output with JOptionPane class

```
import javax.swing.JOptionPane;
public class HelloNameDialog{
  public static void main(String[] args){
      String result;
      result = JOptionPane.showInputDialog("Hay nhap
                                        ten ban:");
      JOptionPane.showMessageDialog(null,
                          "Xin chao "+ result + "!");
      System.exit(0);
  }
}
```



Example on class and object in Java

Class declaration: *each class* is, by default, an *extension of* **Object** (can be omitted)

Class constructor: *initialises* the various *fields*

Class method: *retrieves* and/or *modifies* the *state* of the class

Class fields: **private** means they *can not be accessed* from *outside* the class

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

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

    public void setTime (int h, int m, int s) {
        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
        minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
        second = ( ( s >= 0 && s < 60 ) ? s : 0 );
    }
}
```

Java: Program with object

```
public class Test {

    public static void main (String args[]) {
        Time time = new Time();

        time.hour = 7;
        time.minute = 15;
        time.second = 30;
    }
}
```

```
Test.java:6: hour has private access in Time
      time.hour = 7;
           ^
Time.java:7: minute has private access in Time
      time.minute = 15;
           ^
Time.java:8: second has private access in Time
      time.second = 30;
           ^
3 errors
```


Quiz:

- What is the main different between Procedural Programming and OOP?

123

Procedural Programming – Object Oriented Programming

- Procedural Programming:
  - Program units are procedures or functions.
  - Gap between data and functions
- Object Oriented Programming
  - Program units are objects
  - Data associeted with methods in an object.
  - Each data structure has methods operating on it.

124

Discussion

- Lập trình hướng đối tượng giải quyết tốt hơn những vấn đề nào của lập trình cấu trúc?
- Đọc tài liệu, đưa ra ví dụ minh họa.

Exercises

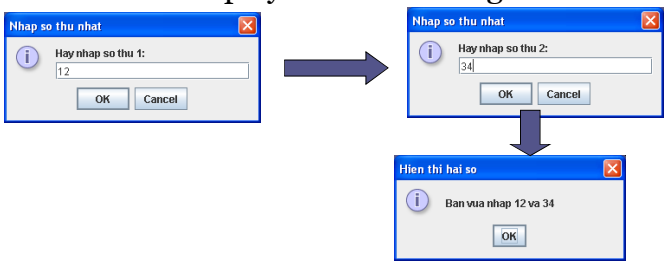
1. Run example programs in Textpad và Jcreator/Netbean/Eclipse.
2. Write a program displaying on the screen two first letters of your name. For example: Thủy → Display TH:

```
TTTTTTTT    H       H
   T        H       H
   T        HHHHHHHH
   T        H       H
   T        H       H
```

3. Study the story of the birth of Java language. (Author, context…)

Exercises(2)

4. Write a program that ask for input 2 integer number and display them on dialogs



5. Modify above program, display Sum, Difference, Product, Quotient of 2 inputted numbers.