Bài 1
TỔNG QUAN VỀ KỸ THUẬT LẬP TRÌNH

Trịnh Thành Trung
trungtt@soict.hust.edu.vn

Nội dung môn học

Chương 1 : Tổng quan về KTLT

Chương 2 : Vài kiến thức nâng cao về C và C++

Chương 3 : Các kỹ thuật viết code hiệu quả và phong cách lập trình

Chương 4 : Một số cấu trúc dữ liệu và giải thuật căn bản

Chương 5 : Các kỹ thuật bẫy lỗi và lập trình phòng ngừa

Chương 6: Testing

Chương 7 : Code turning và documentation

1

TỔNG QUAN

Tổng quan về kỹ thuật lập trình

- Kỹ thuật lập trình là: Kỹ thuật thực thi một giải pháp phần mềm (cấu trúc dữ liệu + giải thuật) dựa trên nền tảng một phương pháp luận (methodology) và một hoặc nhiều ngôn ngữ lập trình phù hợp với yêu cầu đặc thù của ứng dụng.
- Kỹ thuật lập trình
  = Tư tưởng thiết kế+ Kỹ thuật mã hóa
  = Cấu trúc dữ liệu + Giải thuật + Ngôn ngữ lập trình
- Kỹ thuật lập trình ≠ Phương pháp phân tích & thiết kế(A&D)

Thế nào là lập trình?

KHÔNG PHẢI LẬP TRÌNH

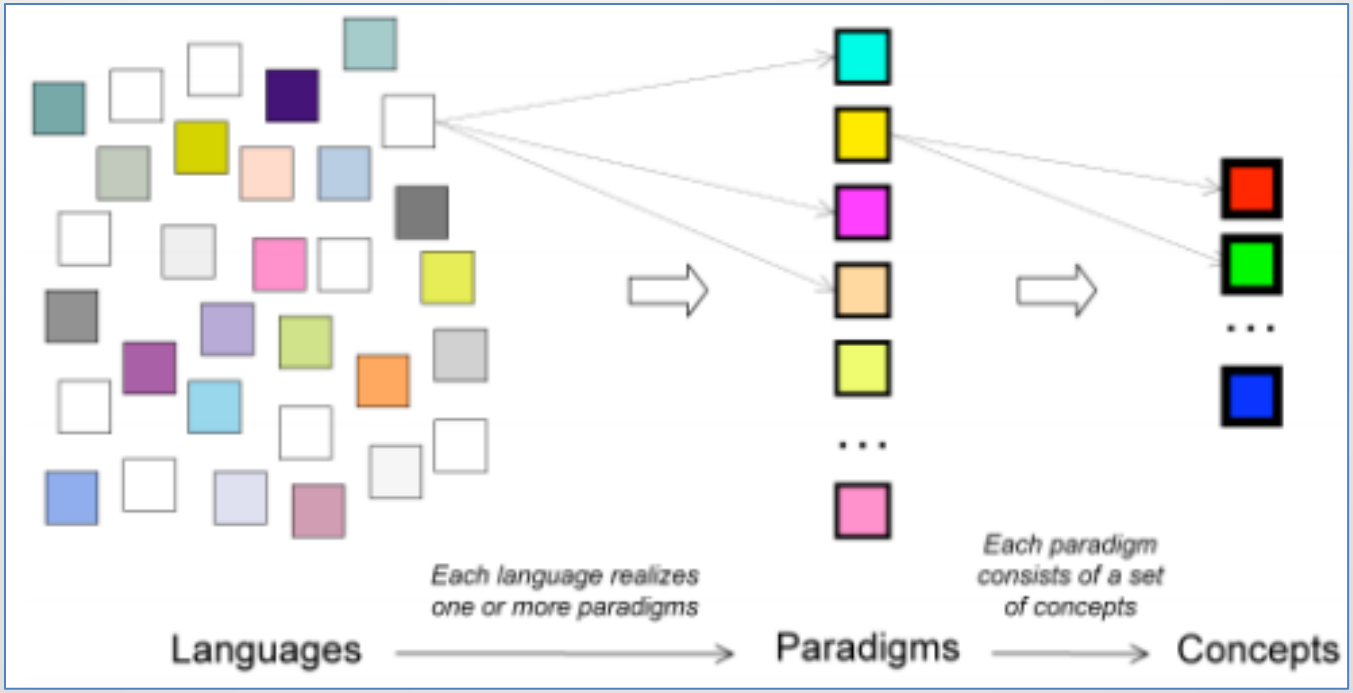
Viết một hàm tính giai thừa!

Viết chương trình in ra N số nguyên tố đầu tiên!

Lập trình giải bài toán: "Vừa gà vừa chó, vừa vặn X con, bó lại cho tròn, đủ Y chân chẵn"

Thế nào là lập trình

- **Với mỗi bài toán (vấn đề) đặt ra, cần:**
  - Thiết kế giải thuật để giải quyết bài toán đó
  - Cài đặt giải thuật bằng một chương trình máy tính



Thế nào là lập trình tốt?

- **Đúng / Chính xác**
  - Thoả mãn đúng các nhiệm vụ bài toán lập trình đặt ra, được khách hàng chấp nhận
- **Ổn định và bền vững**
  - Chương trình chạy ổn định trong mọi trường hợp
  - Chạy ít lỗi (số lượng lỗi ít, cường độ lỗi thấp)
  - Mức độ lỗi nhẹ có thể chấp nhận được
- **Khả năng chỉnh sửa**
  - Dễ dàng chỉnh sửa trong quá trình sử dụng và phát triển
  - Dễ dàng thay đổi hoặc nâng cấp để thích ứng với điều kiện bài toán lập trình thay đổi
- **Khả năng tái sử dụng**
  - Có thể được sử dụng hoặc được kế thừa cho các bài toán #

Thế nào là lập trình tốt?

- Độ tương thích
  - Khả năng thích ứng và chạy tốt trong các điều kiện môi trường khác nhau
- Hiệu suất
  - Chương trình nhỏ gọn, sử dụng ít bộ nhớ
  - Tốc độ nhanh, sử dụng ít thời gian CPU
- Hiệu quả:
  - Thời gian lập trình ngắn,
  - Khả năng bảo trì dễ dàng
  - Giá trị sử dụng lại lớn
  - Sử dụng đơn giản, thân thiện
  - Nhiều chức năng tiện ích

Làm thế nào để lập trình tốt?

- **Học cách tư duy và phương pháp lập trình**
  - Tư duy toán học, tư duy logic, tư duy có cấu trúc, tư duy hướng đối tượng, tư duy tổng quát
  - Tìm hiểu về cấu trúc dữ liệu và giải thuật
- **Hiểu sâu về máy tính**
  - Tương tác giữa CPU, chương trình và bộ nhớ
  - Cơ chế quản lý bộ nhớ
- **Nắm vững ngôn ngữ lập trình**
  - Biết rõ các đặc thù, các khả năng và hạn chế của ngôn ngữ
  - Kỹ năng lập trình (đọc thông, viết thạo)
- **Tự rèn luyện trên máy tính**
  - Hiểu sâu được các điểm nêu trên, Rèn luyện kỹ năng lập trình
  - Thúc đẩy sáng tạo

Các nguyên tắc cơ bản

- Trừu tượng hóa
  - Chắt lọc ra những yếu tố quan trọng, bỏ qua những chi tiết phụ
- Đóng gói
  - Che giấu và bảo vệ các dữ liệu quan trọng qua một giao diện có kiểm soát
- Module hóa
  - Chia nhỏ đối tượng/vấn đề thành nhiều module nhỏ để dễ can thiệp và giải quyết
- Phân cấp
  - Phân hạng hoặc sắp xếp trật tự đối tượng theo các quan hệ trên dưới

Nguyên tắc tối cao

"Keep it simple:

as simple as possible,

but no simpler!"

(Albert Einstein)

Một số khái niệm

- Programming paradigm
  - Là 1 khuôn mẫu - pattern dùng như một Mô hình lập trình máy tính
  - Là 1 mô hình cho 1 lớp các NNLT có cùng những đặc trưng cơ bản
- Programming technique
  - Liên quan đến các ý tưởng thuật toán để giải quyết một lớp vấn đề tương ứng
  - Ví dụ: 'Divide and conquer' và 'program development by stepwise refinement'
- Programming style
  - Là cách chúng ta trình bày trong 1 computer program
  - Phong cách tốt giúp cho chương trình dễ hiểu, dễ đọc, dễ kiểm tra -> dễ bảo trì, cập nhật, gỡ rối, tránh bị lỗi
- Programming culture
  - Tổng hợp các hành vi lập trình, thường liên qua đến các dòng ngôn ngữ lập trình
  - Là tổng thể của Mô hình chính, phong cách và kỹ thuật lập trình
  - Là nhân cách đạo đức trong lập trình cũng như khai thác các CT

2

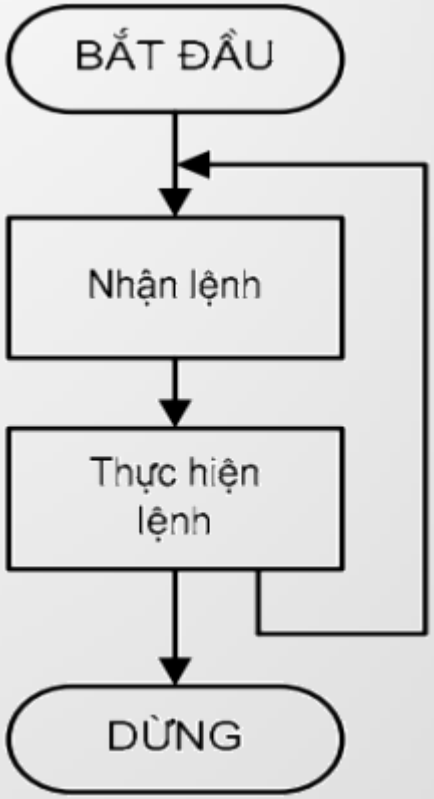
NGÔN NGỮ LẬP TRÌNH

Chương trình máy tính và ngôn ngữ lập trình

- **Computer program**
  - Tập hợp các lệnh chỉ dẫn cho máy tính thực hiện nhiệm vụ
- **Programming language**
  - Dùng để viết các lệnh, chỉ thị

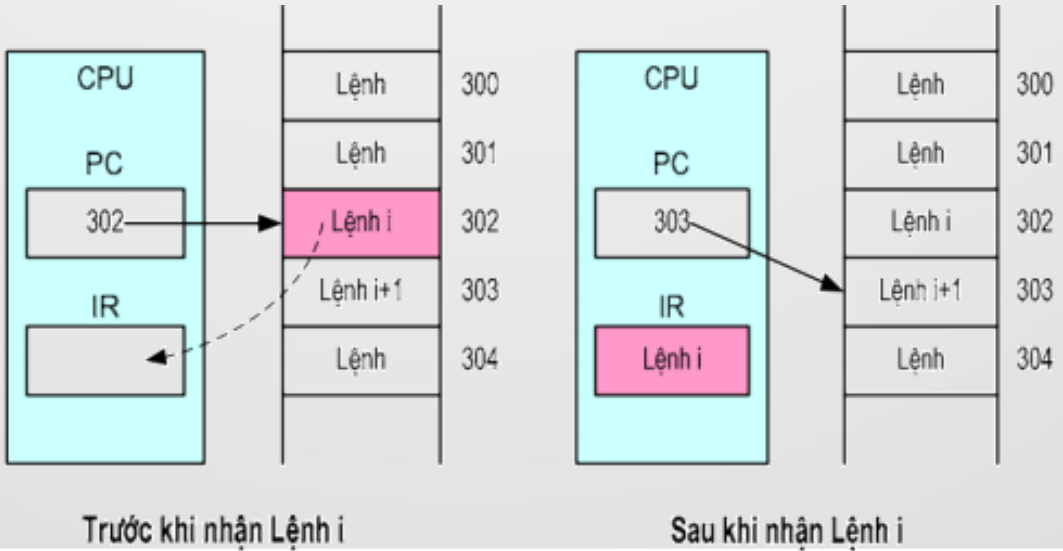
Chương trình máy tính

- Chương trình máy tính được nạp vào bộ nhớ chính (primary memory) như là một tập các lệnh viết bằng ngôn ngữ mà máy tính hiểu được (dãy tuần tự các số nhị phân)
- Tại một thời điểm bất kỳ, máy tính ở một trạng thái (state) nhất định
  - Con trỏ lệnh (instruction pointer) trỏ tới lệnh tiếp theo để thực hiện
- Thứ tự các nhóm lệnh được gọi là luồng điều khiển (control flow)



Chương trình máy tính

- Bắt đầu chu trình lệnh, CPU nhận lệnh từ bộ nhớ chính
  - PC (Program Counter): Thanh ghi địa chỉ của lệnh được nhận
  - Lệnh sau đó được nạp vào thanh ghi lệnh IR (Instruction Register)
- Sau khi lệnh được nhận vào, nội dung PC tự động tăng để trỏ sang lệnh kế tiếp



Ngôn ngữ lập trình

- Một NNLT *là 1 hệ thống các ký hiệu dùng để liên lạc , trao đổi 1 nhiệm vụ / thuật toán với máy tính, làm cho nhiệm vụ được thực thi*. Nhiệm vụ được thực thi gọi là một *computation*, nó tuân thủ một độ chính xác và những quy tắc nhất quán.

- Có rất nhiều NNLT (~1000 NNLT) – phần lớn là các ngôn ngữ hàn lâm, có mục đích riêng hay phát triển bởi 1 tổ chức để phục vụ cho bản thân họ.

Ngôn ngữ lập trình

- Các thành phần cơ bản của ngôn ngữ lập trình
  - Mô thức - Language paradigm là những nguyên tắc chung cơ bản, dùng bởi LTV để xây dựng chương trình.
  - Cú pháp - Syntax của ngôn ngữ là cách để xác định cái gì là hợp lệ trong cấu trúc các câu của ngôn ngữ
  - Ngữ nghĩa - Semantics của 1 program trong ngôn ngữ ấy. Không có semantics, 1 NNLT sẽ chỉ là 1 mớ các câu lệnh vô nghĩa => Semantics là 1 thành phần không thể thiếu của 1 ngôn ngữ.

Ngôn ngữ lập trình

- **Về cơ bản, chỉ có 4 mô hình NNLT chính:**
  - Imperative (Procedural) Paradigm (Fortran, Pascal, C, Ada, ...)
  - Object-Oriented Paradigm (SmallTalk, Java, C++)
  - Logic Paradigm (Prolog)
  - Functional Paradigm (Lisp, ML, Haskell)

Ngôn ngữ lập trình

- **Những tính chất cần có với các chương trình phần mềm:**
  - Tính mềm dẻo scalability / Khả năng chỉnh sửa modifiability
  - Khả năng tích hợp integrability / Khả năng tái sử dụng reusability
  - Tính chuyển đổi, linh hoạt, độc lập phần cứng - portability
  - Hiệu năng cao -performance
  - Độ tin cậy - reliability
  - Dễ xây dựng
  - Rõ ràng, dễ hiểu
  - Ngắn gọn, xúc tích

Mã máy

- Máy tính chỉ nhận các tín hiệu điện tử - có, không có -tương ứng với các dòng bits.
- 1 program ở dạng đó gọi là machine code.
- Ban đầu chúng ta phải dùng machine code để viết CT:
- Quá phức tạp, giải quyết các bài toán lớn là không tưởng

```
23fc 0000 0001 0000 0040
0cb9 0000 000a 0000 0040
6e0c
06b9 0000 0001 0000 0040
60e8
```

Hợp ngữ

- NN Assembly là bước đầu tiên của việc xây dựng cơ chế viết chương trình tiện lợi hơn – thông qua các ký hiệu, từ khóa và cả mã máy.
- Tất nhiên, để chạy được các chương trình này thì phải dịch (assembled) thành machine code.
- Vẫn còn phức tạp, cải thiện không đáng kể

```
   movl        #0x1,n
compare:
   cmpl        #oxa,n
   cgt
        end_of_loop
   acddl       #0x1,n
   bra         compare
end_of_loop:
```

NNLT bậc cao

- Thay vì dựa trên phần cứng (machine-oriented) cần tìm cơ chế dựa trên vấn đề (problem-oriented) để tạo chương trình.
- Chính vì thế NNLT bậc cao – là các ngôn ngữ lập trình gần với ngôn ngữ con người hơn – dùng các từ khóa giống tiếng Anh – đã được xây dựng như : Algol, Fortran, Pascal, Basic, Ada, C, ...

Phân loại theo thời gian

- 1940s : Machine code
- 1950s Khai thác sức mạnh của MT: Assembler code, Autocodes, first version of Fortran
- 1960s Tăng khả năng tính toán: Cobol, Lisp, Algol 60, Basic, PL/1 --- nhưng vẫn dùng phong cách lập trình cơ bản của assembly language.
- 1970s Bắt đầu cuộc khủng hoảng phần mềm “software crisis"
    - Giảm sự phụ thuộc vào máy – Tính chuyển đổi.
    - Tăng sự đúng đắn của CT -Structured Programming, modular programming và information hiding.
    - Ví dụ : Pascal, Algol 68 and C.

Phân loại theo thời gian (tiếp)

- 1980s Giảm sự phức tạp – object orientation, functional programming.
- 1990s Khai thác phần cứng song song và phân tán (parallel và distributed) làm cho chương trình chạy nhanh hơn, kết quả là hàng loạt ngôn ngữ mở rộng khả năng lập trình parallel cũng như các NNLT chuyên parallel như occam được xd.
- 2000s Genetic programming languages, DNA computing, bio-computing?
- Trong tương lai : Ngôn ngữ lt lượng tử: Quantium ?

Các thế hệ NNLT

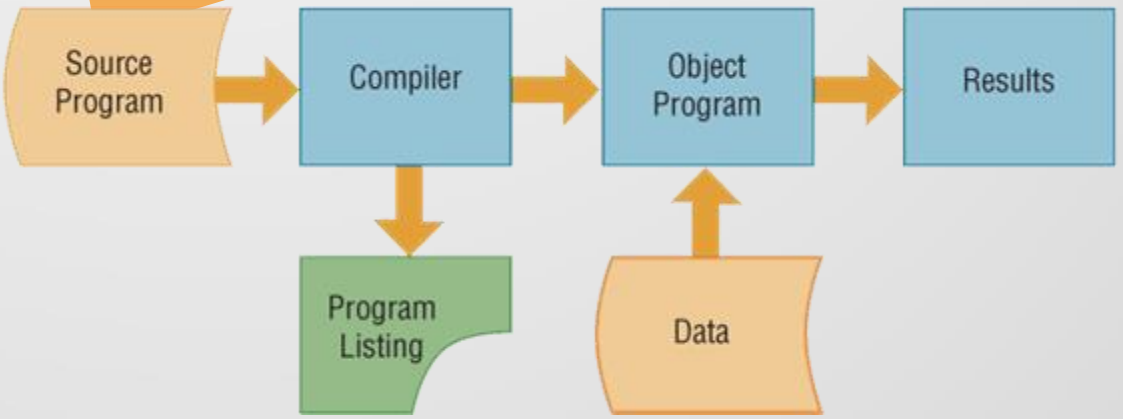
	Classification
1st	Machine languages
2nd	Assembly languages
3rd	Procedural languages
4th	Application languages (4GLs)
5th	AI techniques, inference languages
6th	Neural networks (?), others….

Trình dịch

Trình dịch - Compiler?

- Là chương trình thực hiện biên dịch toàn bộ chương trình nguồn thành mã máy trước khi thực hiện

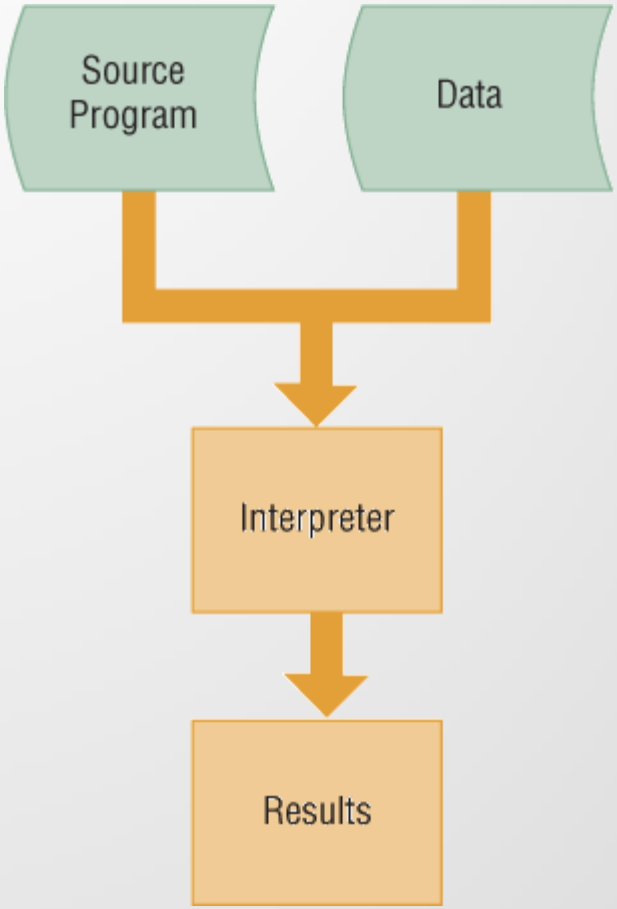
```
*    COMPUTE REGULAR TIME PAY
     MULTIPLY REGULAR-TIME-HOURS BY HOURLY-PAY-RATE
          GIVING REGULAR-TIME-PAY.

*    COMPUTE OVERTIME PAY
     IF OVERTIME-HOURS > 0
          COMPUTE OVERTIME-PAY = OVERTIME-HOURS * 1.5 * HOURLY-PAY-RATE
     ELSE
          MOVE 0 TO OVERTIME-PAY.

*    COMPUTE GROSS PAY
     ADD REGULAR-TIME-PAY TO OVERTIME-PAY
          GIVING GROSS-PAY.

*    PRINT GROSS PAY
     MOVE GROSS-PAY TO GROSS-PAY-OUT.
     WRITE REPORT-LINE-OUT FROM DETAIL-LINE
          AFTER ADVANCING 2 LINES.
```



Thông dịch

Thông dịch - Interpreter?

- Là chương trình dịch và thực hiện từng dòng lệnh của chương trình cùng lúc
- Không tạo ra object program



Ngôn ngữ lập trình hướng đối tượng

- **Object-oriented programming (OOP) language?**

Dùng để hỗ trợ thiết kế HĐT object-oriented design

Object là phần tử chứa đựng cả dữ liệu và các thủ tục xử lý dữliệu

Lợi ích cơ bản là khả năng tái sử dụng - reuse existing objects

Event-driven — Hướng sự kiện Kiểm tra để trả lời một tập các sự kiện

Event là hành động mà chương trình cần đáp ứng

C++ và Java là các NN hoàn toàn HĐT

object-oriented languages

C++

- Chứa đựng các thành phần của C, loại bỏ những nhược điểm và thêm vào những tính năng mới để làm việc với các nguyên lý hướng đối tượng.

```
// portion of a C++ program that allows users to create a new zip code from a
// string or a number and expand zip codes, as appropriate, to a 10-digit number

ZipC::ZipC( const unsigned long zipnum )
{
  ostringstream strInt;
  strInt << zipnum;
  code = strInt.str();
}

const string ZipC::getCode()
{
  return code;
}

void ZipC::setCode(const string newCode)
{
  code = newCode;
}

void ZipC::expand( const string suffix )
{

  if(code.length() == 5 &&       // small size?
     suffix.length() == 4)       // length ok?
  {
     code += "-";
     code.append(suffix);
  }
}
```

Java

- Phát triển bởi Sun Microsystems
- Giống C++ nhưng dùng trình dịch just-in-time (JIT) để chuyển source code thành machine code

```
public void actionPerformed(ActionEvent e)
   {
      foundKey = false;

      //Search for the key pressed
      for (int i = 0; i < keysArray.length && !foundKey; i++)
         if(e.getSource() == keysArray[i])    //key match found
         {
            foundKey = true;
            switch(i)
            {
               case 0: case 1: case 2: case 3: case 4:  //number buttons
               case 5: case 6: case 7: case 8: case 9:  //0 - 9
               case 15:                                 //decimal point button
                  if(clearText)
                  {
                     lcdField.setText("");
                     clearText = false;
                  }
                  lcdField.setText(lcdField.getText() + keysArray[i].getLabel());
                  break;
```

Ngôn ngữ lập trình trực quan

Visual programming language

Visual programming environment (VPE) Cho phép developers kéo và thả các objects để xd programs

Cung cấp giao diện trực quan hoặc đồ họa để tạo source code

Đôi khi được gọi là fifth-generation language

LTV viết và phát triển chương trình trong các segments

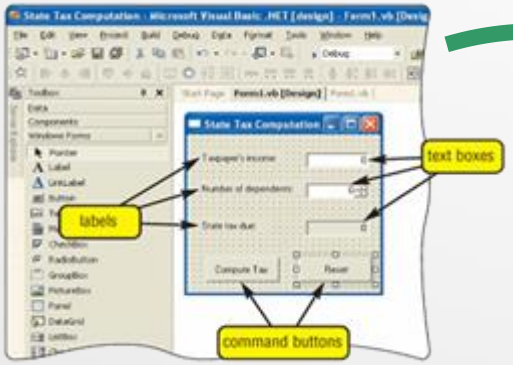
Thường được dùng trong môi trường RAD (rapid application development)

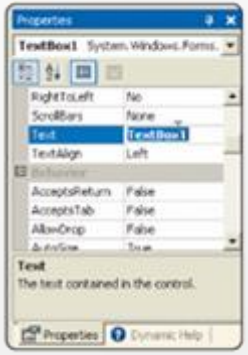
Ngôn ngữ lập trình trực quan

- Visual Studio
  - Bước phát triển của visual programming languages và RAD tools
  - .NET là platform cho nhiều ngôn ngữ lập trình của Microsoft như C#, Basic hay C++
  - Dùng để xây dựng các ứng dụng Windows cũng như phát triển ứng dụng web

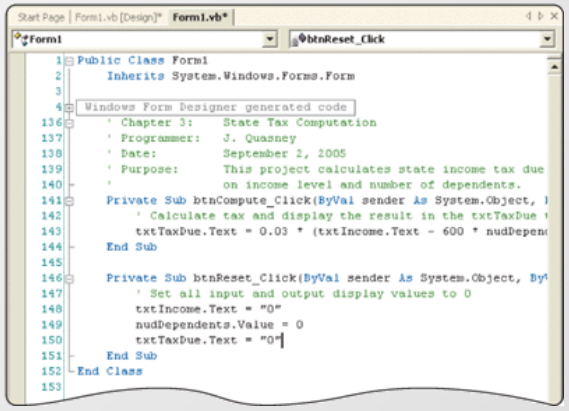
Ngôn ngữ lập trình trực quan

Step 1. LTV thiết kế giao diện người dùng - user interface.





Step 2. LTV gán các thuộc tính cho mỗi object trên form.

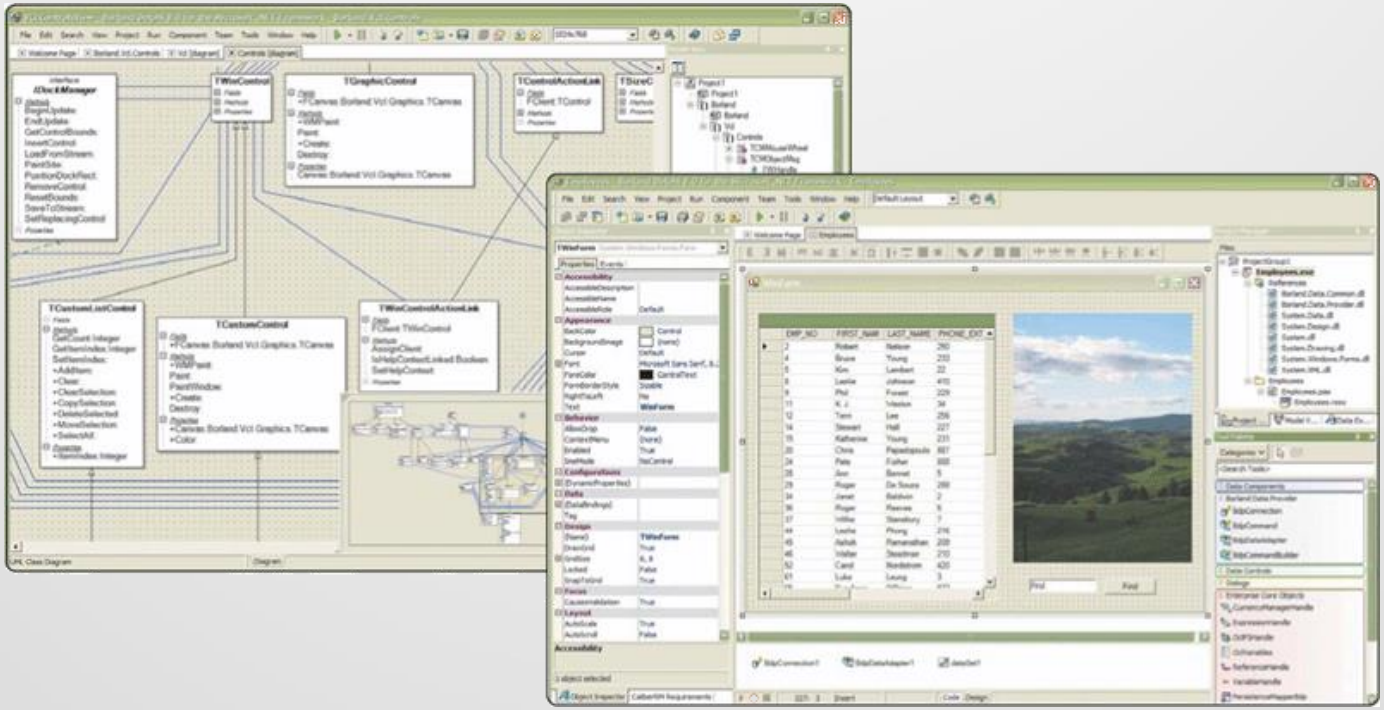


Step 3. LTV viết code để xác định các action cần thực hiện đối với các sự kiện cần thiết.

Step 4. LTV kiểm tra application.

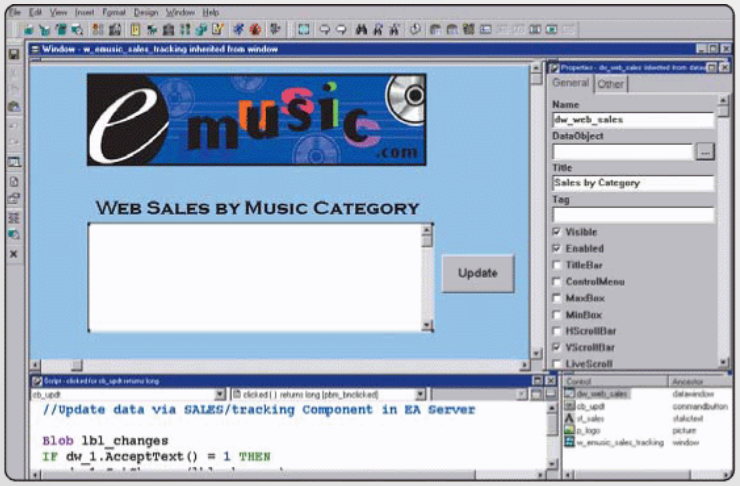
Ngôn ngữ lập trình trực quan

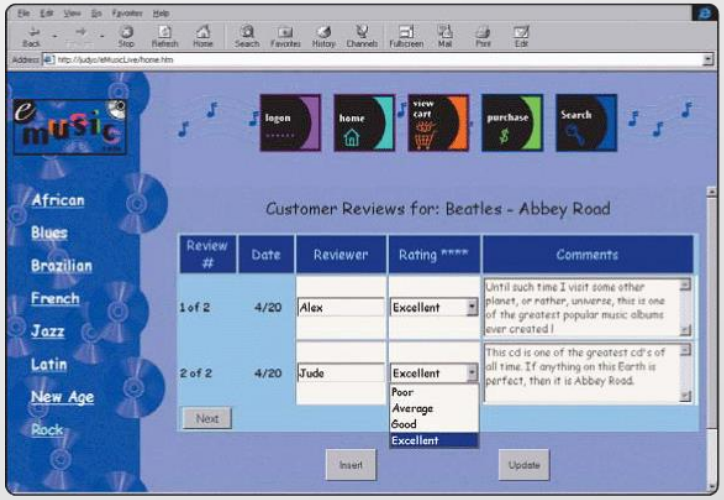
- **Delphi**
  - Là 1 công cụ lập trình trực qua mạnh
  - Hợp với những ứng dụng chuyên nghiệp và Web lớn



Ngôn ngữ lập trình trực quan

- **PowerBuilder**
  - Một công cụ lập trình trực quan mạnh khác
  - Phù hợp với các ứng dụng Web-based hay các ứng dụng lớn HĐT - object-oriented applications





Ngôn ngữ phi thủ tục và công cụ phát triển phần mềm

Nonprocedural Language

LTV viết các lệnh giống tiếng Anh hoac tương tác với môi trường trực quan để nhận được các dữ liệu từ files hay database

Program Development Tools

Các chương trình thân thiện với người sử dụng được thiết kế để trợ giúp cả LTV & người sử dụng trong việc tạo chương trình

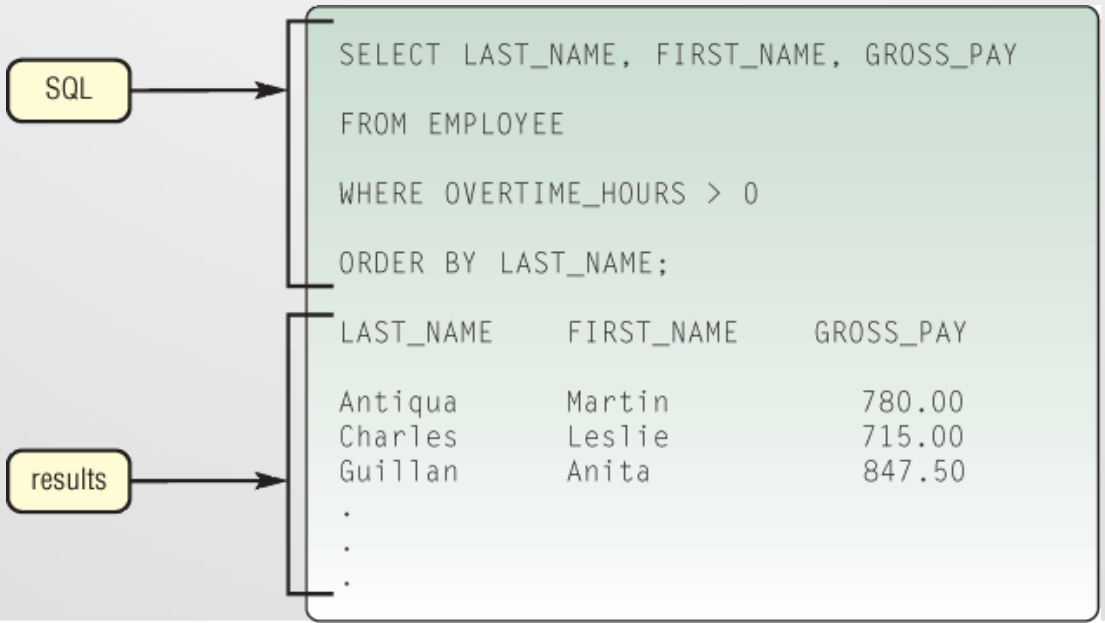
RPG (Report Program Generator)

- Các ngôn ngữ LT phi thủ tục được dùng để tạo các báo cáo, thiết lập các thao tác tính toán và cập nhật files

```
C* COMPUTE REGULAR TIME PAY
C           RTHRS     MULT RATE            RTPAY     72
C*
C* COMPUTE OVERTIME PAY
C           OTHRS     IFGT 0
C           RATE      MULT 1.5             OTRATE    72
C           OTRATE    MULT OTHRS           OTPAY     72
                      ELSE
C                     INZ                  OTPAY     72
C
C* COMPUTE GROSS PAY
C           RTPAY     ADD  OTPAY           GRPAY     72
C
C* PRINT GROSS PAY
C                     EXCPTDETAIL
C
C*
O* OUTPUT SPECIFICATIONS
OQPRINT  E            DETAIL
O                                    23 'THE GROSS PAY IS $'
O                     GRPAY  J       34
```

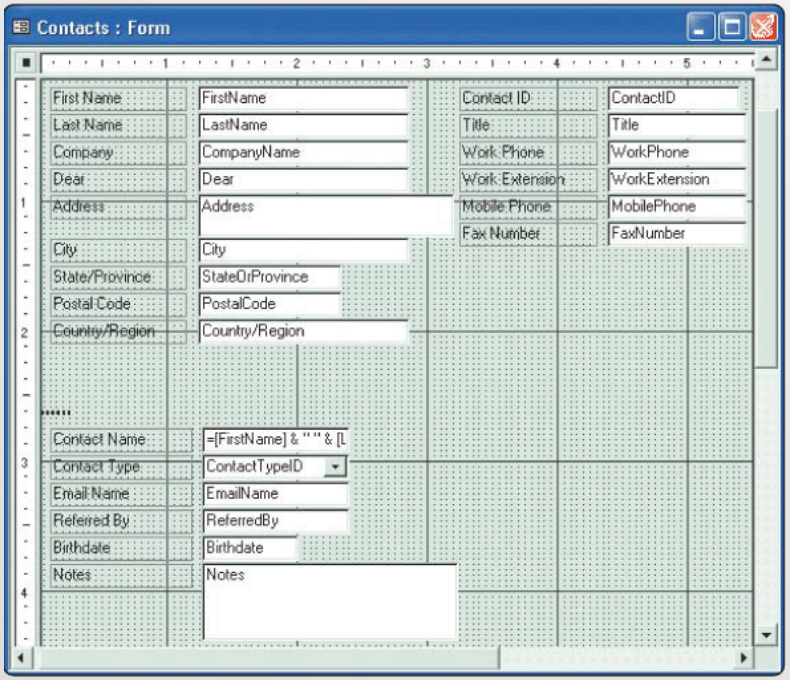
4GL (4th generation language)

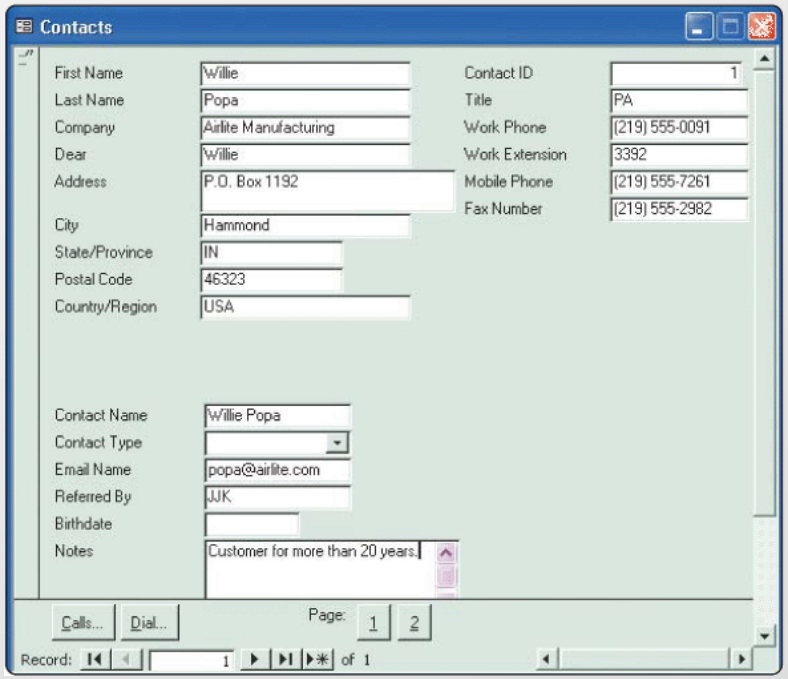
- Là các ngôn ngữ phi thủ tục cho phép truy cập dữ liệu trong CSDL
- NNLT 4GL thông dụng là SQL, Access, là các ngôn ngữ truy vấn. Cho phép users quản trị dữ liệu trong csdl quan hệ relational DBMS



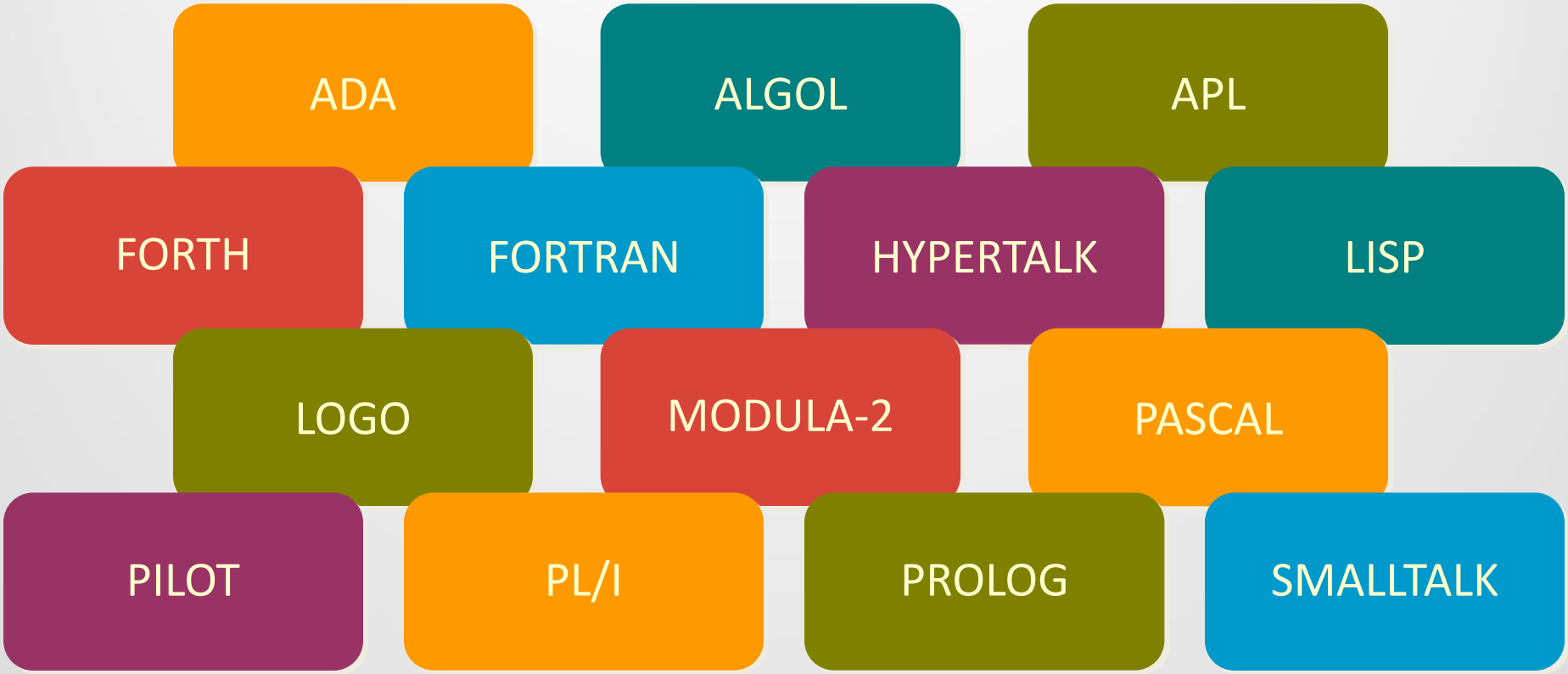
Application Generator

- Là chương trình tạo source code hoặc machine code từ các specification
- Bao gồm các chương trình tạo Report , form, và tạo menu





Các Programming Languages khác



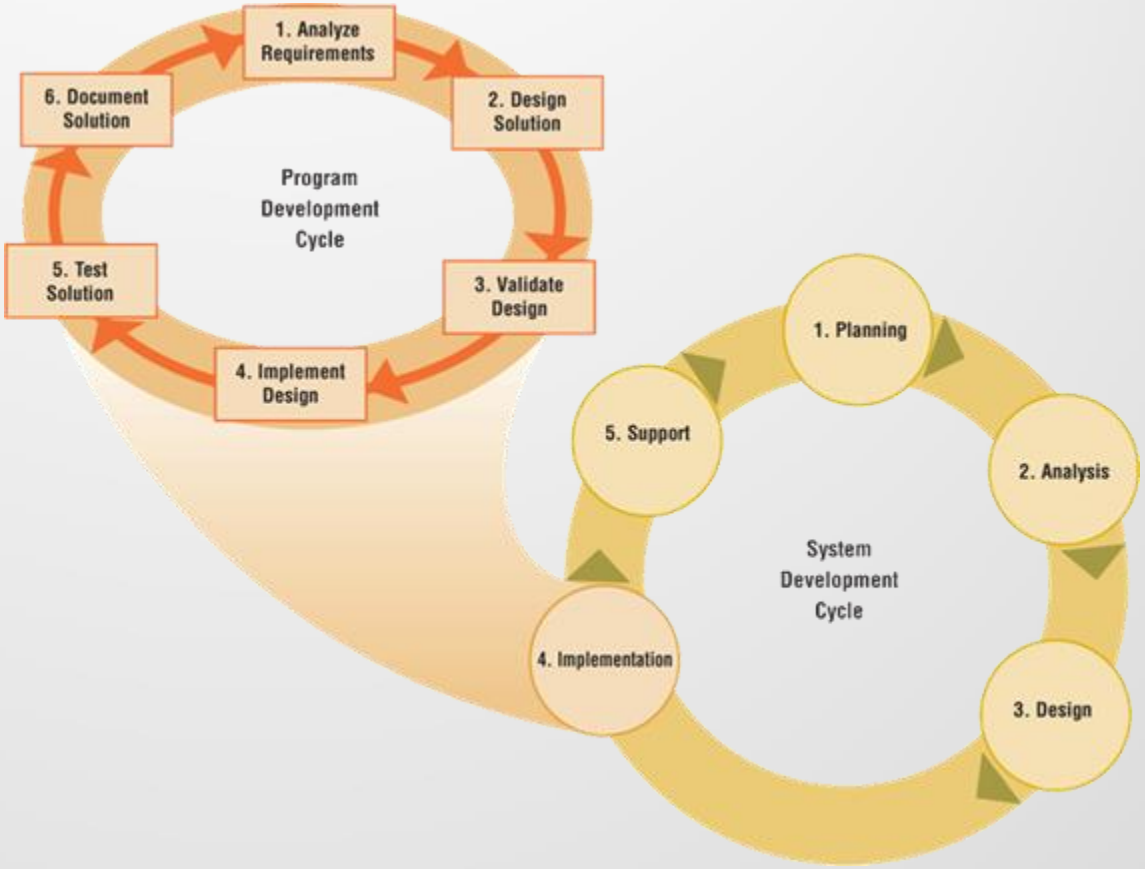
3

CHU TRÌNH PHÁT TRIỂN PHẦN MỀM

Chu trình phát triển phần mềm

Program development cycle?

- Là các bước mà LTV dùng để xây dựng phần mềm



Step 1 — Analyze Requirements

- **Các việc cần làm khi phân tích yêu cầu?**
  - Thiết lập các requirements
  - Gặp các nhà phân tích hệ thống và users
  - Xác định input, output, processing, và các thành phần dữ liệu

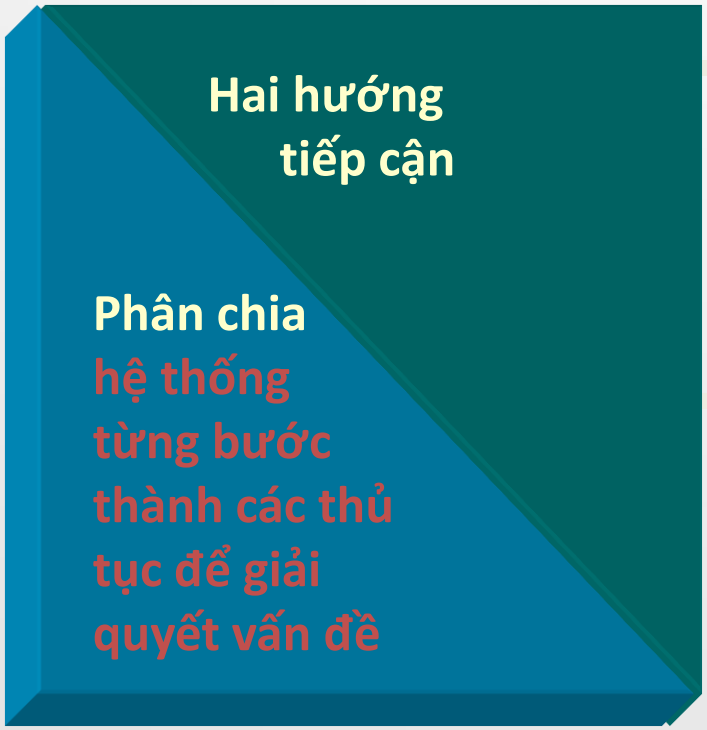
IPO CHART

Input	Processing	Output
Regular Time Hours Worked	Read regular time hours worked, overtime hours worked, hourly pay rate.	Gross Pay
Overtime Hours Worked	Calculate regular time pay.	
Hourly Pay Rate	If employee worked overtime, calculate overtime pay.	
	Calculate gross pay.	
	Print gross pay.	

IPO chart—Xác định đầu vào, đầu ra và các bước xử lý

Step 2 — Design Solution

- Thiết kế giải pháp?



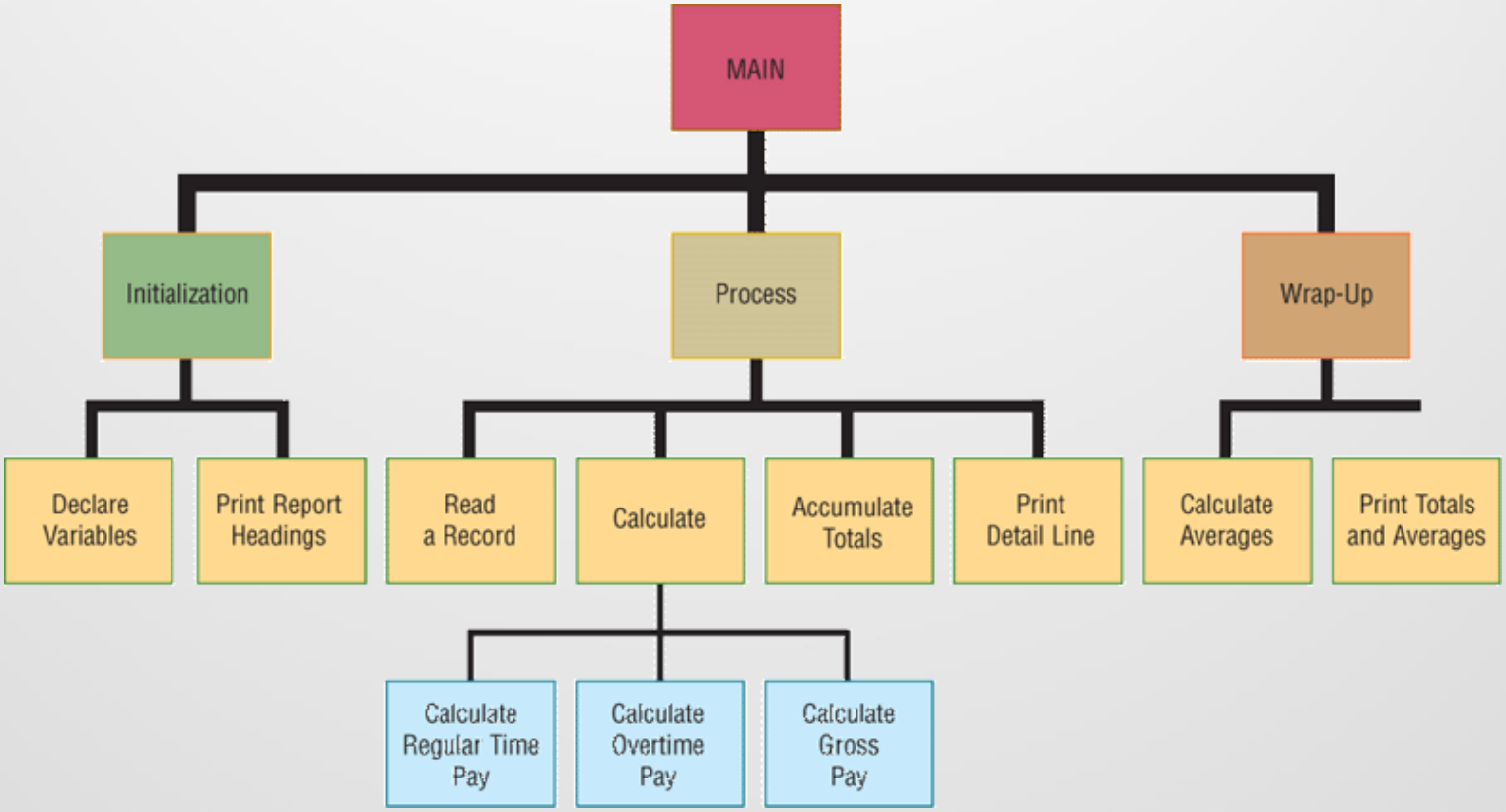
Object-oriented design

Structured design, còn gọi là top-down design

LTV bắt đầu với thiết kế Tổng thể rồi đi đến thiết kế chi tiết

Step 2 — Design Solution (tiếp)

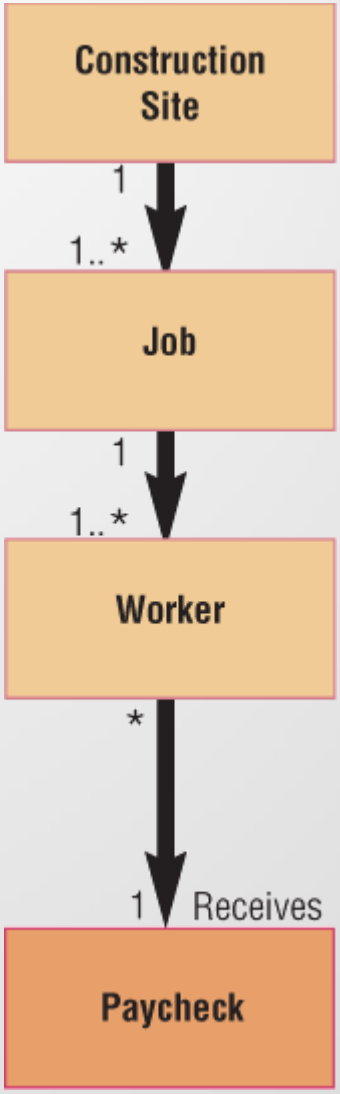
- **Sơ đồ phân cấp chức năng- hierarchy chart**
  - Trực quan hóa các modules chương trình
  - Còn gọi là sơ đồ cấu trúc



Step 2 — Design Solution (tiếp)

Object-oriented (OO) design

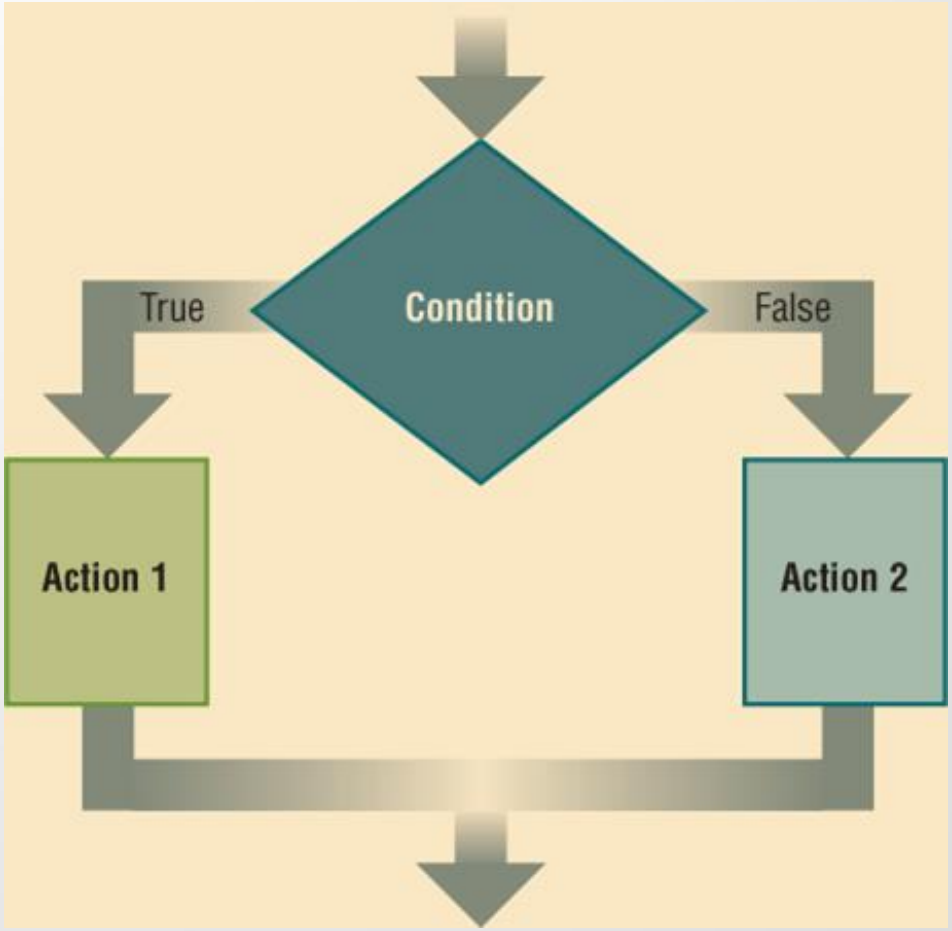
- LTV đóng gói dữ liệu và các thủ tục xử lý dữ liệu trong 1 object
- Các objects được nhóm lại thành các class
- Biểu đồ lớp thể hiện trực quan các quan hệ phân cấp quan hệ của các class



p. 689 Fig. 13-27

Step 2 — Design Solution (tiếp)

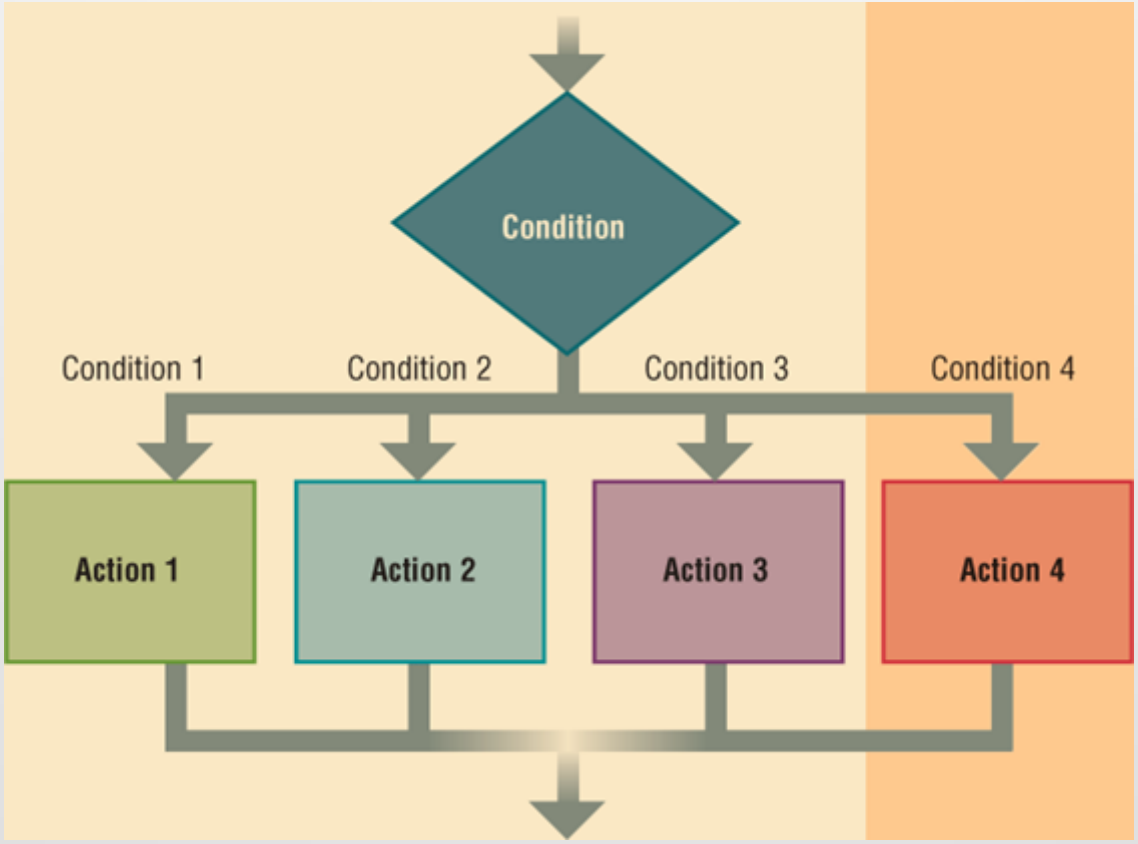
- Cấu trúc tuyển chọn
- Chỉ ra action tương ứng điều kiện
- 2 kiểu
  - Case control structure
  - If-then-else control structure—dựa theo 2 khả năng: true or false



p. 690 Fig. 13-29

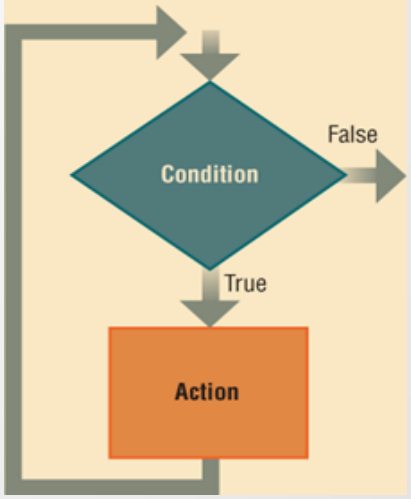
Step 2 — Design Solution (tiếp)

- Case control structure
- Dựa theo 3 hoặc nhiều hơn các khả năng

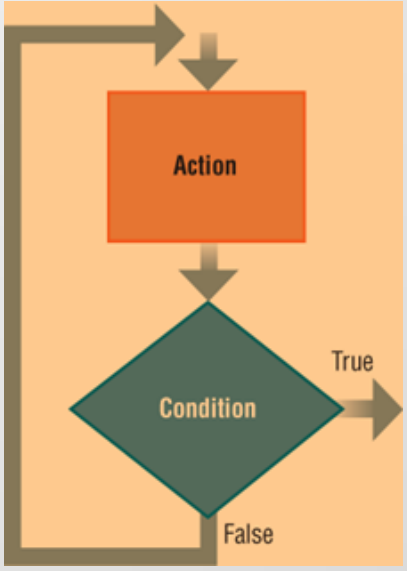


Step 2 — Design Solution (tiếp)

- Cấu trúc lặp
- Cho phép CT thực hiện 1 hay nhiều actions lặp đi lặp lại
- Do-while control structure—lặp khi điều kiện còn đúng
- Do-until control structure—Lặp cho đến khi điều kiện đúng



Do-While Control Structure



Do-Until Control Structure

Step 3 – Validate Design

- **Các tác vụ:**

Kiểm tra
độ chính xác
của program

Desk check
LTV dùng các dữ liệu
thử nghiệm để kiểm tra ct

Test data
các dữ liệu thử nghiệm
giống như số liệu thực mà
CT sẽ thực hiện

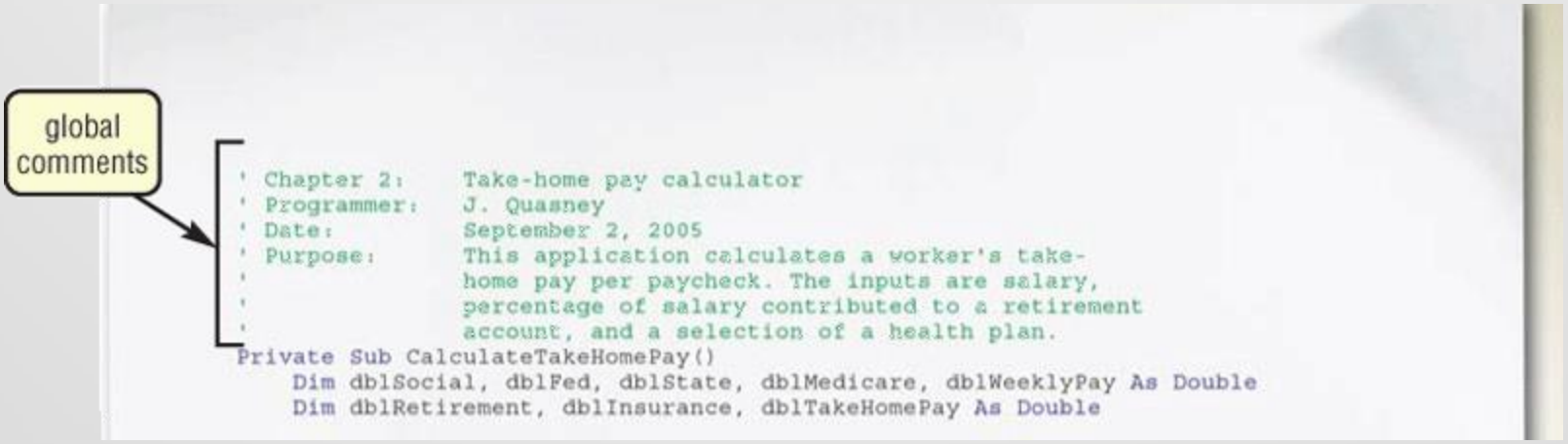
LTV kiểm tra
logic cho tính đúng đắn
và thử tìm các lỗi logic

Logic error
các sai sót khi thiết kế
gây ra những kết quả
không chính xác

Structured walkthrough
LTV mô tả logic
của thuật toán trong khi
programming team duyệt theo
logic chương trình

Step 4 — Implement Design

- **Viết code : dịch từ thiết kế thành program**
  - Syntax—Quy tắc xác định cách viết các lệnh
  - Comments—program documentation
  - Extreme programming (XP)—coding và testing ngay sau khi các yêu cầu được xác định



Step 5 – Test Solution

- **Các tác vụ:**

Đảm bảo CT chạy thông và cho kết quả chính xác

Debugging - Tìm và sửa các lỗi syntax và logic errors

Kiểm tra phiên bản beta, giao cho Users dùng thử và thu thập phản hồi

Step 6 — Document Solution

- **Các tác vụ:**

Rà soát lại program code—loại bỏ các **dead code**, tức các lệnh mà chương trình không bao giờ gọi đến

Rà soát, hoàn thiện documentation

Tóm tắt

Có hàng loạt các NNLT dùng để viết computer programs

Chu trình phát triển chương trình và các công cụ được dùng để làm cho quá trình này hiệu quả hơn

4 mô hình lập trình cơ bản

4

CÁC MÔ THỨC LẬP TRÌNH

Mô thức lập trình cơ bản

- **Bốn mô thức lập trình cơ bản**
  - Imperative paradigm
  - Functional paradigm
  - Logical paradigm
  - Object-oriented paradigm
- **Bên cạnh đó có thể kể đến**
  - Visual paradigm
  - Parallel paradigms
- **Một vài mô thức mới khác**
  - Concurrent programming
  - Distributed programming
  - Extreme programming

Imperative paradigm

- Với Mô hình này ý tưởng cơ bản là các lệnh gây ảnh hưởng đáng kể đến trạng thái chương trình.
- Mỗi imperative program bao gồm
  - Declarative statements – các lệnh khai báo, chúng cung cấp các tên cho biến. Các biến này có thể thay đổi giá trị trong quá trình thực hiện Chương trình.
  - Assigment statements – Lệnh gán: gán giá trị mới cho biến
  - Program flow control statements – Các lệnh điều khiển cấu trúc chương trình: Xác định trình tự thực hiện các lệnh trong chương trình.
  - Module : chia ct thành các chương trình con: Functions & Procedures

Imperative paradigm (tiếp)

- Về mặt nguyên lý và ý tưởng: Công nghệ phần cứng digital và ý tưởng của Von Neumann
- Các bước tính toán, thực hiện với mục đích kiểm soát cấu trúc điều khiển. Chúng ta gọi các bước là các mệnh lệnh - commands
- Tương ứng với cách mô tả các công việc hàng ngày như là trình tự nấu ăn hay sửa chữa xe cộ
- Những lệnh đặc trưng của imperative languages là : Assignment, IO, procedure calls
- Các ngôn ngữ đại diện: Fortran, Algol, Pascal, Basic, C
- Các thủ tục và hàm chính là hình ảnh về sự trừu tượng: che giấu các lệnh trong CT con, có thể coi CT con là 1 lệnh
- Còn gọi là "Procedural programming"

Functional paradigm

- Functional programming trên nhiều khía cạnh là đơn giản và rõ ràng hơn imperative. Vì nguồn gốc của nó là toán học thuần túy: Lý thuyết hàm. Trong khi imperative paradigm bắt nguồn từ ý tưởng công nghệ cơ bản là digital computer, phức tạp hơn, kém rõ ràng hơn lý thuyết toán học về hàm.
- Functional programming dựa trên nền tảng khái niệm toán học về hàm và 1 NNLT hàm bao gồm ít nhất những thành phần sau :
    - Tập hợp các cấu trúc dữ liệu và các hàm liên quan
    - Tập hợp các hàm cơ sở - Primitive Functions.
    - Tập hợp các toán tử .

Functional paradigm (tiếp)

- Về mặt nguyên lý và ý tưởng: Toán học và lý thuyết hàm
- Các giá trị tạo được là không thể biến đổi non-mutable
- Không thể thay đổi các yếu tố của giá trị hợp thành
- Giống như phương thuốc, có thể tạo một phiên bản của các giá trị hợp thành : một giá trị trung gian
- Trừu tượng 1 biểu thức đơn thành 1 hàm và hàm có thể tính toán như là 1 biểu thức
- Các hàm là những giá trị đầu tiên
- Hàm là dữ liệu hoàn chỉnh, giống như số, danh sách, ...
- Thích hợp với xu hướng tính toán theo yêu cầu
- Mở ra những khả năng mới

Ví dụ về Functional programming

```
long GT(long n) {
        long  x=1;
        while (n > 0) {
                x *=  n;
                n - ;
        }
        return x;
}
```

```
long GT( long n) {
    if (n==1 ) return 1;
    else  return  n* GT(n-1);
}
```

Với functional paradigm, ta có thể viết

GT n =

if n = 1 then 1

else n * GT(n 1);

Logic Paradigm

- Mô hình lập trình logic hoàn toàn khác với các mô hình còn lại.
- Mô hình này đặc biệt phù hợp với những lĩnh vực liên quan đến việc rút ra những kiến thức từ những sự kiện và quan hệ cơ bản – lĩnh vực trí tuệ nhân tạo. Có vẻ như mô hình này không gắn với những lĩnh vực tính toán nói chung.
- Trả lời 1 câu hỏi thông qua việc tìm các giải pháp
- Các đặc trưng:
  - Về nguyên tắc và ý tưởng: Tự động kiểm chứng trong trí tuệ nhân tạo
  - Dựa trên các chân lý- tiên đề axioms,các quy luật suy diễn - inference rules, và các truy vấn queries.
  - Chương trình thực hiện từ việc tìm kiếm có hệ thống trong 1 tập các sự kiện, sử dụng 1 tập các luật để đưa ra kết luận

Object-oriented Paradigm

- Mô hình hướng đối tượng thu hút được sự quan tâm và nổi tiếng từ khoảng 20 năm nay. Lý do là khả năng hỗ trợ mạnh của tính bao gói và gộp nhóm logic của các khía cạnh lập trình. Những thuộc tính này rất quan trọng khi mà kích cỡ các chương trình ngày càng lớn.
- Nguyên nhân cơ bản và sâu sắc dẫn đến thành công của mô hình này là:
  - Cơ sở lý thuyết đỉnh cao của mô hình. 1 CT HĐT được xây dựng với những khái niệm, tư tướng làm cơ sở, điều đó rất quan trọng và theo cách đó tất cả các kỹ thuật cần thiết cho lập trình trở thành thứ yếu.
- Gủi thông điệp giữa các objects để mô phỏng sự tiến triển theo thời gian của hàng loạt các hiện tượng trong thế giới thực

Object-oriented Paradigm

- Các đặc trưng
  - Nguyên lý và ý tưởng : Lý thuyết về concepts, và các mô hình tương tác trong thế giới thực
  - Dữ liệu cũng như các thao tác trên dữ liệu được đóng gói trong objects
  - Cơ chế che dấu thông tin được sử dụng để tránh những tác động từ bên ngoài object
  - Các Objects tương tác với nhau qua việc truyền thông điệp, đó là phép ẩn dụ cho việc thực hiện các thao tác trên 1 object
  - Trong phần lớn các NNLT HĐT, objects được nhóm lại trong classes
    - Objects trong classes có chung các thuộc tính, cho phép lập trình trên lớp, thay vì lập trình trên từng đối tượng riêng lẻ
    - Classes đại diện cho concepts còn objects đại diện cho hiện tượng
    - Classes được tổ chức trong cây phả hệ có kế thừa
    - Tính kế thừa cho phép mở rộng hay chuyên biệt hóa lớp