Bài 9
XÂY DỰNG TÀI LIỆU

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

XÂY DỰNG TÀI LIỆU

1. Các tài liệu trong và ngoài (internal và external)
2. Các quy tắc xây dựng tài liệu
3. Quy ước viết mã nguồn và chú thích

1

CÁC LOẠI TÀI LIỆU CHƯƠNG TRÌNH

Các loại tài liệu chương trình

- Tài liệu trong (Internal documentation)
  - Các chú thích cho mã nguồn (comments)
- Tài liệu ngoài (External documentation)
  - Dành cho các lập trình viên khác khi làm việc với mã nguồn
- Tài liệu dành cho người sử dụng
  - Cẩm nang dành cho những người sử dụng mã nguồn
- Các loại tài liệu khác
  - Tài liệu kiểm thử…

2

TÀI LIỆU TRONG

Chú thích hợp lý

- Các chú thích có giúp người đọc hiểu được mã nguồn hay không ?
- Các chú thích có thực sự bổ ích hay không? Lập trình viên viết chú thích vì chú thích là thực sự cần thiết để hiểu mã nguồn hay viết để cho có ?
- Người đọc có dễ dàng làm việc với mã nguồn hơn khi có chú thích hay không?
- Nếu không chú thích được thì nên đặt tham chiếu đến một tài liệu cho phép người đọc hiểu vấn đề sâu hơn.

Các cách chú thích cần tránh: Chú thích vô nghĩa

- **Làm chủ ngôn ngữ**
  - Hãy để chương trình tự diễn tả bản thân
  - Rồi…
- **Viết chú thích để thêm thông tin**

```
i= i+1; /* Add one to i */

for (i= 0; i < 1000; i++) { /* Tricky bit */
    …
}
int x,y,q3,z4;  /* Define some variables */

int main()
/* Main routine */

while (i < 7) { /*This comment carries on and on */
```

Các chú thích cần tránh: Chú thích chỉ nhằm mục đích phân đoạn mã nguồn

```
while (j < ARRAYLEN) {
    printf ("J is %d\n", j);
    for (i= 0; i < MAXLEN; i++) {
/* These comments only */
        for (k= 0; k < KPOS; k++) {
/* Serve to break up */
            printf ("%d %d\n",i,k);
/* the program */
        }
/* And make the indentation */
    }
/* Very hard for the programmer to see */
    j++;
}
```

Viết bao nhiêu chú thích là đủ ?

- Chú thích là tốt, nhưng điều đó không có nghĩa là dòng lệnh nào cũng cần viết chú thích
- Chú thích các đoạn (“paragraphs") code, đừng chú thích từng dòng
  - vd., “Sort array in ascending order"
- Chú thích dữ liệu tổng thể
  - Global variables, structure type definitions, ….
- Nhiều chú thích quá sẽ làm cho mã nguồn trở nên khó đọc

Viết bao nhiêu chú thích là đủ?

- Ít chú thích quá làm mã nguồn trở nên khó hiểu
- Chỉ viết chú thích nếu trong vòng 1 phút bạn không thể hiểu nổi đoạn lệnh đó làm gì, như thế nào
- Viết chú thích tương ứng với code
  - Thay đổi khi code thay đổi.

Ví dụ

- Chú thích các đoạn ("paragraphs") code, đừng chú thích từng dòng code

```
#include <stdio.h>
#include <stdlib.h>

int main(void)

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0 if successful. */

{
   const double PI = 3.14159;
   int radius;
   int diam;
   double circum;

   /* Read the circle's radius. */
   printf("Enter the circle's radius:\n");
   if (scanf("%d", &radius) != 1)
   {
      fprintf(stderr, "Error: Not a number\n");
      exit(EXIT_FAILURE);  /* or:  return EXIT_FAILURE; */
   }
…
```

Ví dụ

```
    /* Compute the diameter and circumference. */
    diam = 2 * radius;
    circum = PI * (double)diam;

    /* Print the results. */
    printf("A circle with radius %d has diameter %d\n",
       radius, diam);
    printf("and circumference %f.\n", circum);

    return 0;
}
```

Những thành phần nào của mã nguồn bắt buộc phải có chú thích

- Tất cả các file (nếu chương trình gồm nhiều file) đều cần chú thích về nội dung của file đó
- Tất cả các hàm: dùng để làm gì, dùng các biến đầu vào nào, trả ra cái gi.
- Biến có tên không rõ ràng
  - `i,j,k` cho vòng lặp, `FILE *fptr` không cần chú thích
  - nhưng `int total;` cần
- Tất cả các struct/typedef (trừ phi nó thực sự quá tầm thường)

File comments

```
/*************************************************************************
class:  GigaTron (GIGATRON.CPP)
author: Dwight K. Coder
date:   July 4, 2014
Routines to control the twenty-first century's code evaluation
tool. The entry point to these routines is the EvaluateCode()
routine at the bottom of this file.
*************************************************************************/
```

Function Comments

- Mô tả những gì cần thiết để gọi hàm 1 cách chính xác
  - Mô tả hàm **làm gì**, chứ không phải nó **làm như thế nào**
  - Bản thân Code phải rõ ràng, dễ hiểu để biết cách nó làm việc...
  - Nếu không, hãy viết chú thích bên trong định nghĩa hàm
- Mô tả đầu vào: Tham số truyền vào, đọc file gì, biến tổng thể được dùng
- Mô tả outputs: Giá trị trả về, tham số truyền ra, ghi ra file gì, các biến tổng thể mà nó tác động tới

Ví dụ

- Bad

```
/* decomment.c */

int main(void) {

/* Đọc 1 ký tự.  Dựa trên ký tự ấy và trạng thái
DFA hiện thời, gọi hàm xử lý trạng thái tương ứng.
Lặp cho đến hết tệp end-of-file. */

    …
}
```

– Giải thích hàm làm như thế nào

Ví dụ

- Good

```
/* decomment.c */

int main(void) {

/* Đọc 1 chương trình C qua stdin.
   Ghi ra stdout với mỗi chú thích thay bằng 1 dấu
cách.
   Trả về 0 nếu thành công, EXIT_FAILURE nếu không
thành công. */


   …
}
```

  - Giải thích hàm làm gì

Các quy tắc viết chú thích khác

- Chú thích nếu bạn cố tình thực hiện 1 thao tác kỳ cục khiến các LTV khác khó hiểu
- Nếu chú thích quá dài, tốt nhất là nên đặt tham chiếu sang đoạn văn bản mô tả chi tiết ở chỗ khác
- Đừng cố gắng định dạng chú thích theo cách có thể gây nhầm lẫn với mã nguồn (ví dụ, đặt gióng hàng riêng cho chú thích)

3

TÀI LIỆU NGOÀI

Tài liệu ngoài

- Giới thiệu với các LTV khác mã nguồn dùng để làm gì
- Nhiều công ty lớn tự đặt chuẩn riêng để viết tài liệu ngoài
- Mục tiêu là cho phép các LTV khác sử dụng và thay đổi mã nguồn mà không cần đọc và hiểu từng dòng lệnh

Viết tài liệu ngoài: Bước 1

- **Miêu tả một cách tổng quát cách thức hoạt động của chương trình**
  - Chương trình phải làm gì?
  - Phải đọc từ nguồn dữ liệu nào, ghi vào đâu?
  - Giả thiết gì với đầu vào?
  - Dùng giải thuật nào?

Viết tài liệu ngoài: Bước 2

- Miêu tả một cách tổng quát quy trình nghiệp vụ của chương trình (giống như cách miêu tả một flowchart)
- Có thể vẽ biểu đồ
- Giải thích các giải thuật phức tạp được sử dụng trong chương trình, hoặc cho biết có thể tìm được lời giải thích ở đâu

Viết tài liệu ngoài: Bước 3

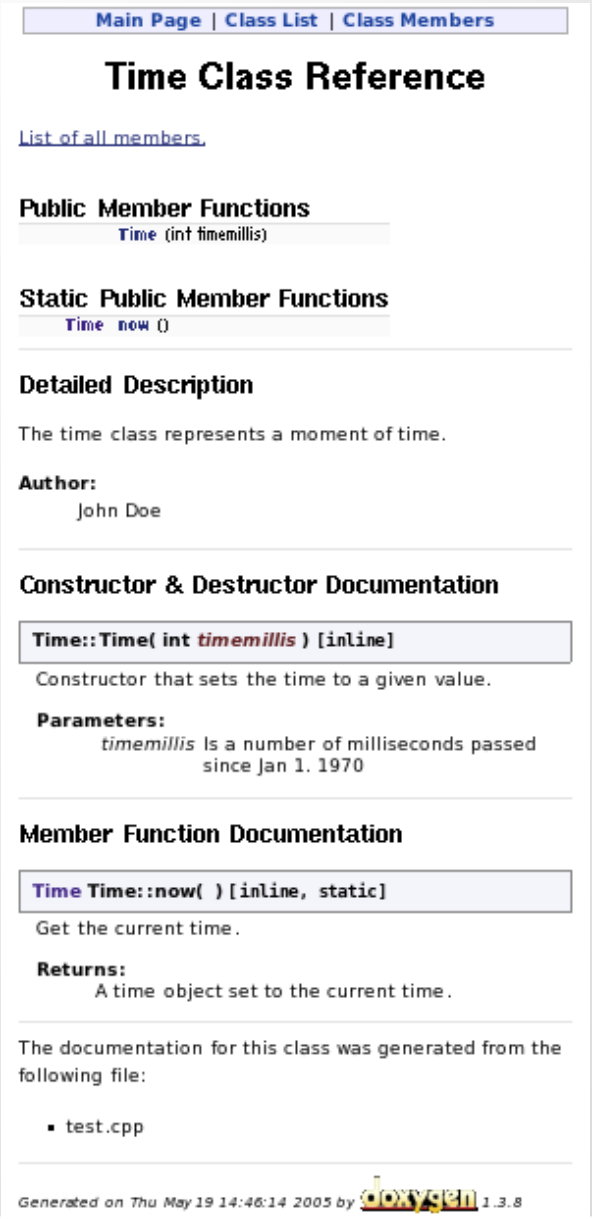
- Nếu chương trình bao gồm nhiều file, giải thích nội dung từng file
- Giải thích cấu trúc dữ liệu được sử dụng phổ biến trong chương trình
- Giải thích việc sử dụng các biến toàn cục trong các chương trình con

Viết tài liệu ngoài: Bước 4

- **Miêu tả các hàm chính trong chương trình**
  - LTV tự quyết định hàm nào là hàm chính trong chương trình của mình
  - Xem xét hàm nào là hàm nghiệp vụ thực sự, không nhất thiết phải là hàm dài nhất hay khó viết nhất
- **Miêu tả các tham số đầu vào và giá trị trả về**

Các công cụ sinh tài liệu

- Doxygen
  - Hỗ trợ nhiều ngôn ngữ lập trình (C/C++, C#, Java, PHP, Python…)
  - Chạy trên Linux, Windows, MacOS



Main Page | Class List | Class Members

Time Class Reference

List of all members.

Public Member Functions

Time (int timemillis)

Static Public Member Functions

Time now ()

Detailed Description

The time class represents a moment of time.

Author:
John Doe

Constructor & Destructor Documentation

Time::Time(int *timemillis*) [inline]

Constructor that sets the time to a given value.

Parameters:
timemillis Is a number of milliseconds passed since Jan 1. 1970

Member Function Documentation

Time Time::now( ) [inline, static]

Get the current time.

Returns:
A time object set to the current time.

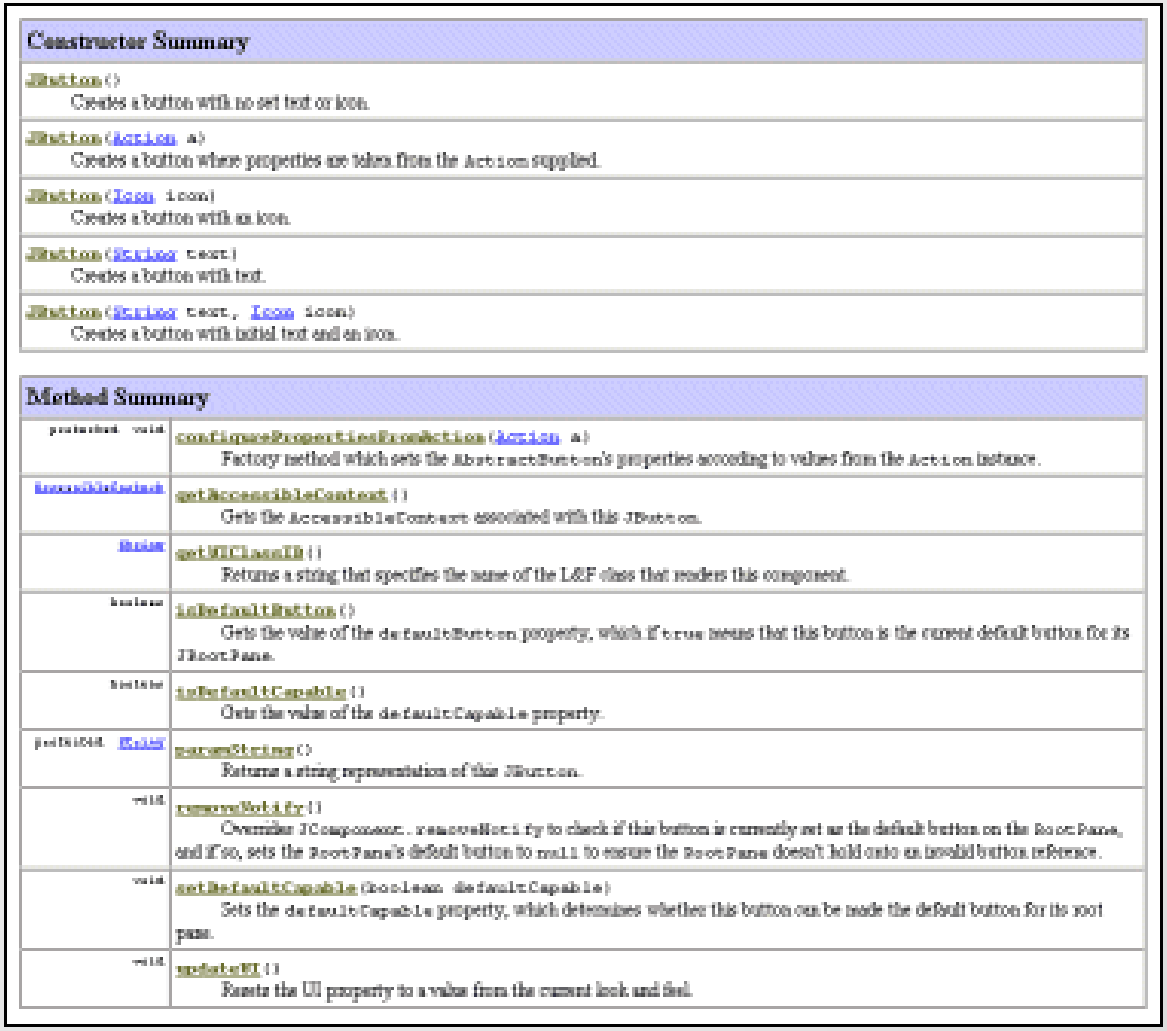
The documentation for this class was generated from the following file:

- test.cpp

Generated on Thu May 19 14:46:14 2005 by doxygen *1.3.8*

Các công cụ sinh tài liệu

- Javadoc
  - Có sẵn trong JavaSDK
  - Một số IDE tự động sinh tài liệu javadoc



Constructor Summary

`JButton()`
Creates a button with no set text or icon.

`JButton(Action a)`
Creates a button where properties are taken from the `Action` supplied.

`JButton(Icon icon)`
Creates a button with an icon.

`JButton(String text)`
Creates a button with text.

`JButton(String text, Icon icon)`
Creates a button with initial text and an icon.

Method Summary

protected void	`configurePropertiesFromAction(Action a)` Factory method which sets the `AbstractButton`'s properties according to values from the `Action` instance.
AccessibleContext	`getAccessibleContext()` Gets the `AccessibleContext` associated with this `JButton`.
String	`getUIClassID()` Returns a string that specifies the name of the L&F class that renders this component.
boolean	`isDefaultButton()` Gets the value of the `defaultButton` property, which if `true` means that this button is the current default button for its `JRootPane`.
boolean	`isDefaultCapable()` Gets the value of the `defaultCapable` property.
protected String	`paramString()` Returns a string representation of this `JButton`.
void	`removeNotify()` Overrides `JComponent.removeNotify` to check if this button is currently set as the default button on the `RootPane`, and if so, sets the `RootPane`'s default button to `null` to ensure the `RootPane` doesn't hold onto an invalid button reference.
void	`setDefaultCapable(boolean defaultCapable)` Sets the `defaultCapable` property, which determines whether this button can be made the default button for its root pane.
void	`updateUI()` Resets the UI property to a value from the current look and feel.

4

TÀI LIỆU NGƯỜI DÙNG VÀ TÀI LIỆU KIỂM THỬ

Viết tài liệu cho người dùng

- Đây chính là hướng dẫn sử dụng (user manual)
- Là phần không thể thiếu khi viết tài liệu cho 1 dự án phần mềm, nhưng không phải phần quan trọng nhất

Viết tài liệu kiểm thử

- Tài liệu kiểm thử là 1 trong số các tài liệu quan trong của 1 dự án phần mềm
- Nếu được, bạn nên viết ra 1 số bằng chứng về việc bạn đã kiểm thử chương trình của bạn với nhiều đầu vào khác nhau
- Việc không viết tài liệu kiểm thử có thể gây ra nhiều hậu quả nặng nề