

**ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH
ĐOÀN KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH
CÂU LẠC BỘ TIN HỌC**



Hướng dẫn thực hành Java và Eclipse

Tác giả: HỒNG TRUNG DŨNG

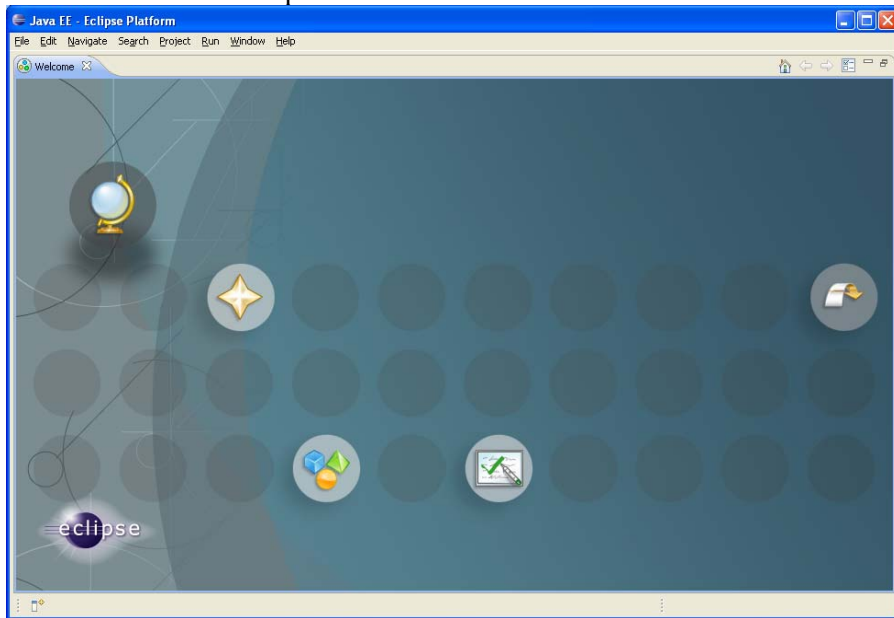
Tháng 5 - 2008

1. Cài đặt Java và Eclipse

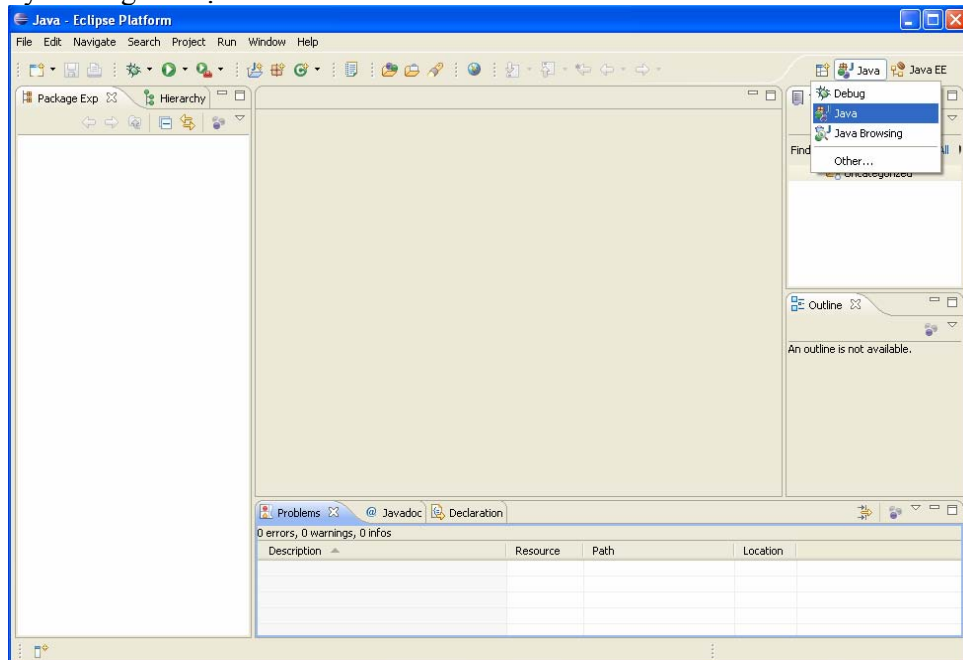
- _ Download JDK (phiên bản 1.5 trở lên) tại www.java.sun.com
- _ Cài đặt JDK theo hướng dẫn
- _ Download Eclipse phiên bản Europa (Java Developers hoặc Java EE Developers) tại www.eclipse.org
- _ Giải nén file eclipse.zip ra một thư mục

2. Thay đổi giao diện

Màn hình Welcome của Eclipse:



Chuyển đổi giao diện ban đầu của Java EE thành Java:

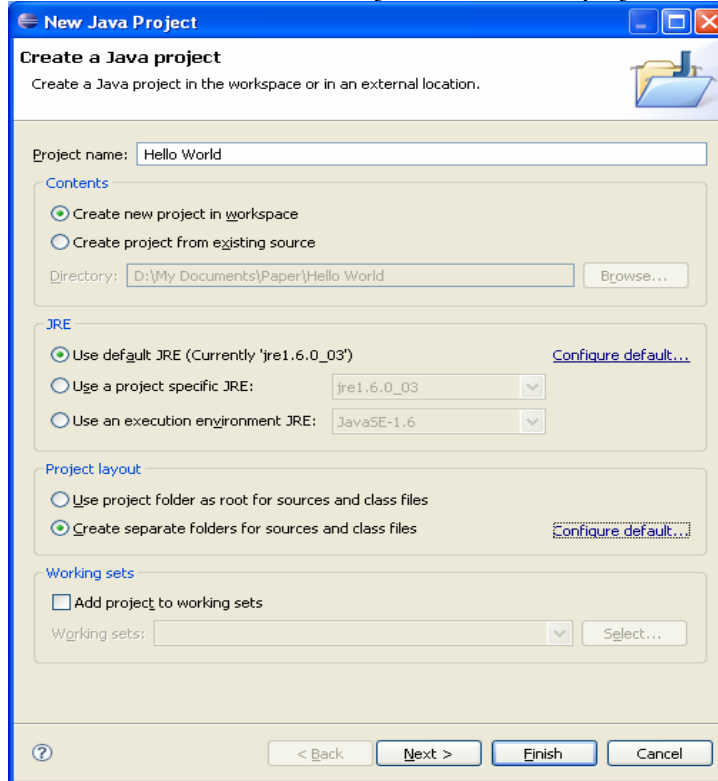


Mở rộng cửa sổ Editor, tự động hide các khung như Package Explorer, Outline đi: click phải vào các khung này, chọn Fast View

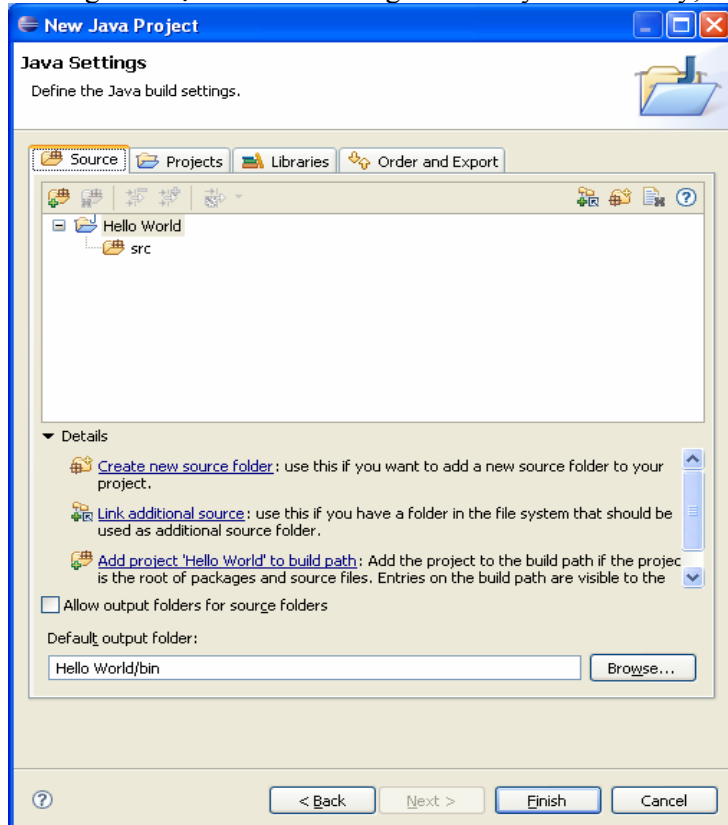
3. Khởi tạo và quản lý Project

a. Tạo một project java

Chọn File → New → Java Project. Đặt tên cho project và chọn Next

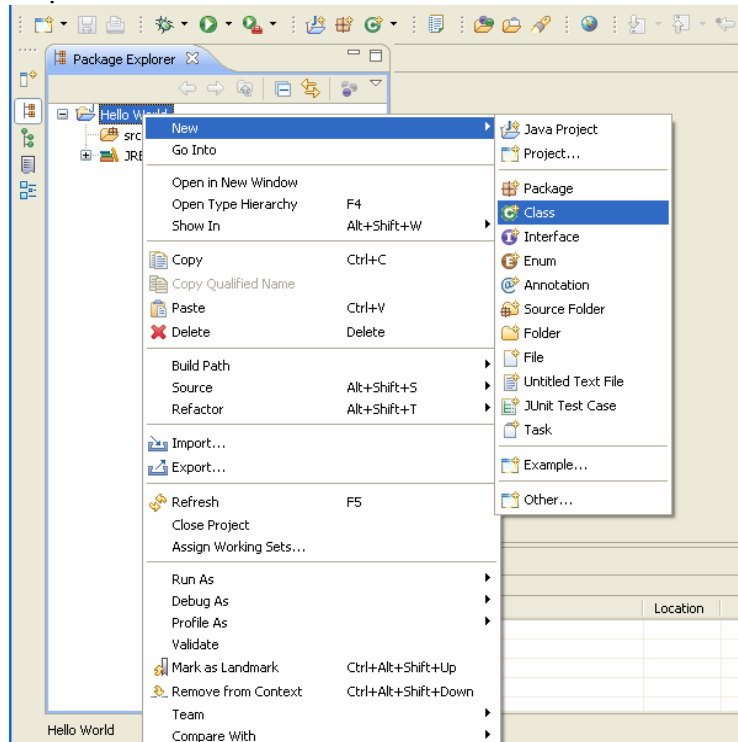


Màn hình tiếp theo: các project java thường để source code trong thư mục src và để các file class trong thư mục bin. Nếu không muốn thay đổi điều này, chọn Next

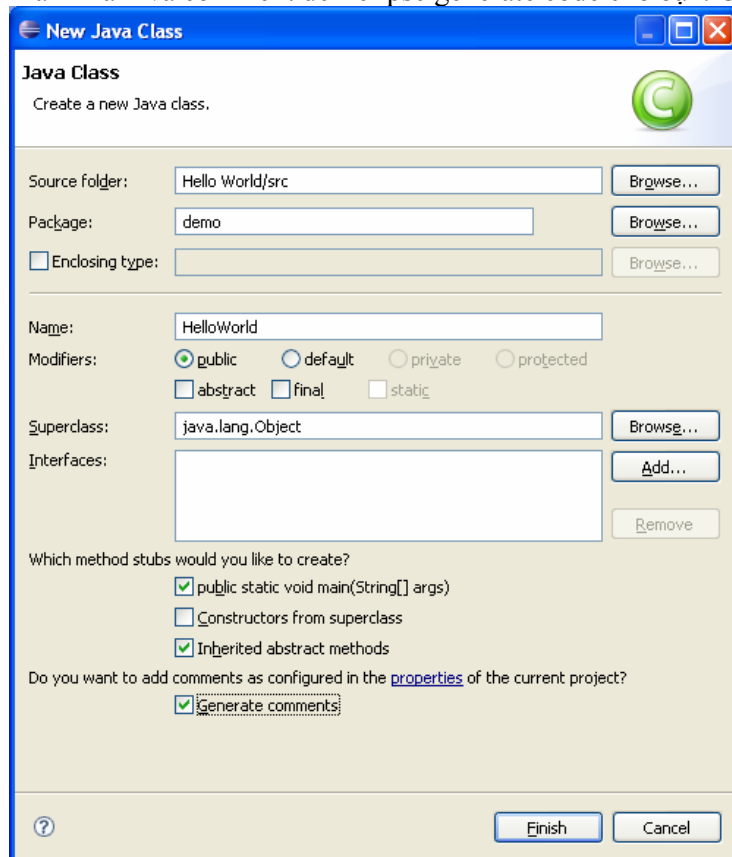


b. Thêm một class mới vào project


Chọn New → Class



Chọn package sẽ chứa class này và điền tên class vào. Theo quy ước, tên package bắt đầu bằng chữ thường, tên class bắt đầu bằng chữ hoa. Sau đó, chọn tiếp các thông số như sinh sẵn hàm main và comment để Eclipse generate code cho bạn. Chọn Finish để kết thúc



Eclipse sẽ tự động tạo lớp mới như sau:



```

HelloWorld.java
+ /**
package demo;

- /**
 * @author Trung Dung
 *
 */
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

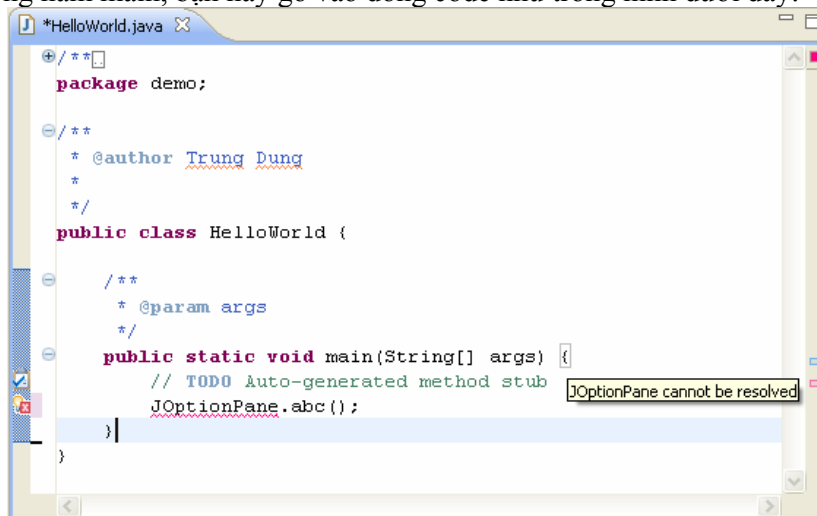
    }

}

```

c. Viết code cho hàm main

Chúng ta sẽ viết một chương trình hiện ra một cửa sổ thông báo mang nội dung tùy chọn. Trong hàm main, bạn hãy gõ vào dòng code như trong hình dưới đây:



```

*HelloWorld.java
+ /**
package demo;

- /**
 * @author Trung Dung
 *
 */
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        JOptionPane.abc();
    }

}

```

Cơ chế dịch tức thời của Eclipse sẽ cho biết chương trình của bạn đang có lỗi, biểu hiện bằng dấu đỏ ở bên phải thanh trượt dọc. Lý do là chương trình của chúng ta chưa import thư viện cần thiết để xử dụng lớp JOptionPane. Điều này cũng giống như ta phải khai báo include cho các chương trình C++. Eclipse có thể giúp chúng ta làm điều này một cách tự động. Hãy nhấn tổ hợp phím Ctrl-Shift-O, Eclipse sẽ tự động tìm kiếm package có chứa lớp JOptionPane và tự động thêm vào. Hình sau cho thấy kết quả sau khi đã thêm dòng import.



```
package demo;

import javax.swing.JOptionPane;

/**
 * @author Trung Dung
 */
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        JOptionPane.abc();
    }
}
```

Chương trình vẫn còn lỗi do phương thức abc không tồn tại trong lớp JOptionPane. Hãy sửa lại đoạn code. Xóa chuỗi “abc” đi, để con trỏ sau dấu ‘.’ và nhấn Ctrl-Space, Eclipse sẽ hiển ra cửa sổ hỗ trợ như sau:



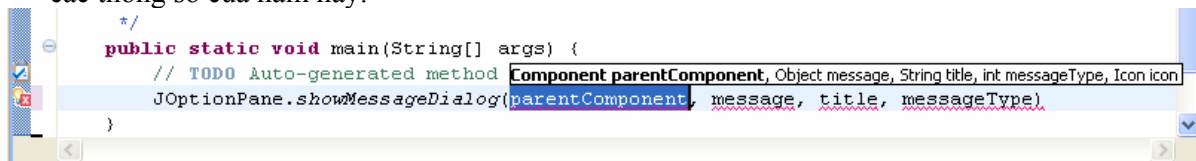
```
package demo;

import javax.swing.JOptionPane;

/**
 * @author Trung Dung
 */
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        JOptionPane.show
    }
}
```

Hãy chọn phương thức showMessageDialog ở gần cuối và Enter. Eclipse sẽ hỗ trợ bạn điền các thông số của hàm này.

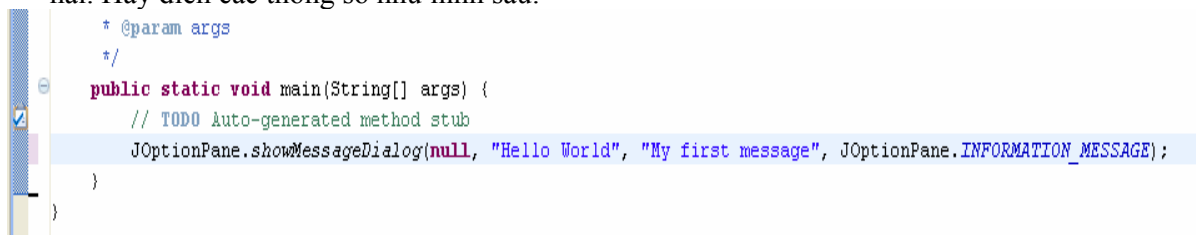


```
package demo;

import javax.swing.JOptionPane;

/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    JOptionPane.showMessageDialog(parentComponent, message, title, messageType);
}
}
```

Hãy đánh vào chuỗi “null” cho thông số thứ nhất và nhấn phím Tab để nhập vào thông số thứ hai. Hãy điền các thông số như hình sau:



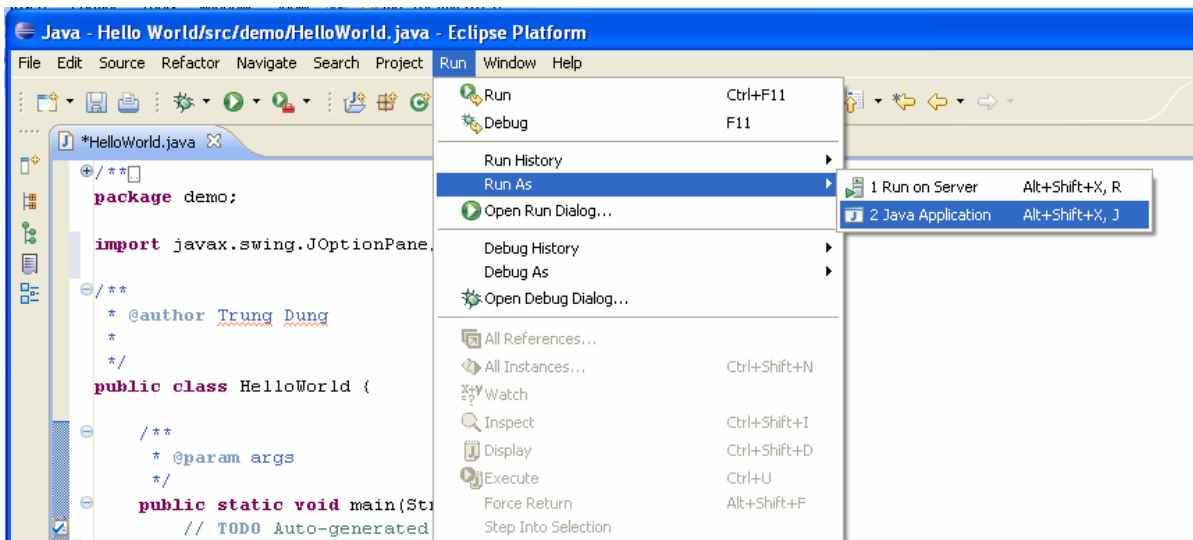
```
package demo;

import javax.swing.JOptionPane;

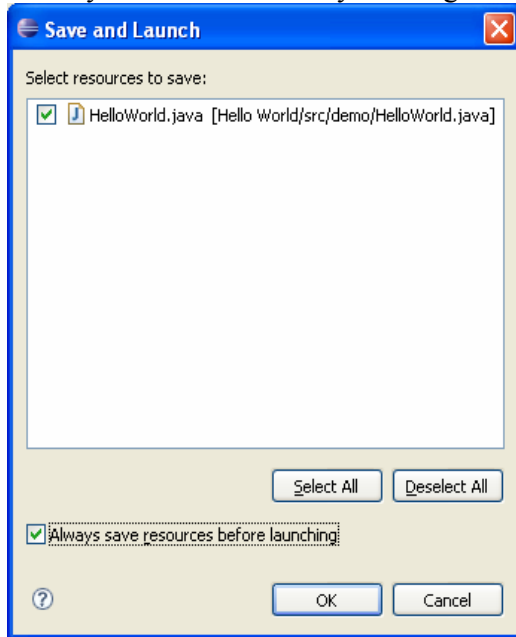
/**
 * @param args
 */
public static void main(String[] args) {
    // TODO Auto-generated method stub
    JOptionPane.showMessageDialog(null, "Hello World", "My first message", JOptionPane.INFORMATION_MESSAGE);
}
}
```

d. Chạy chương trình

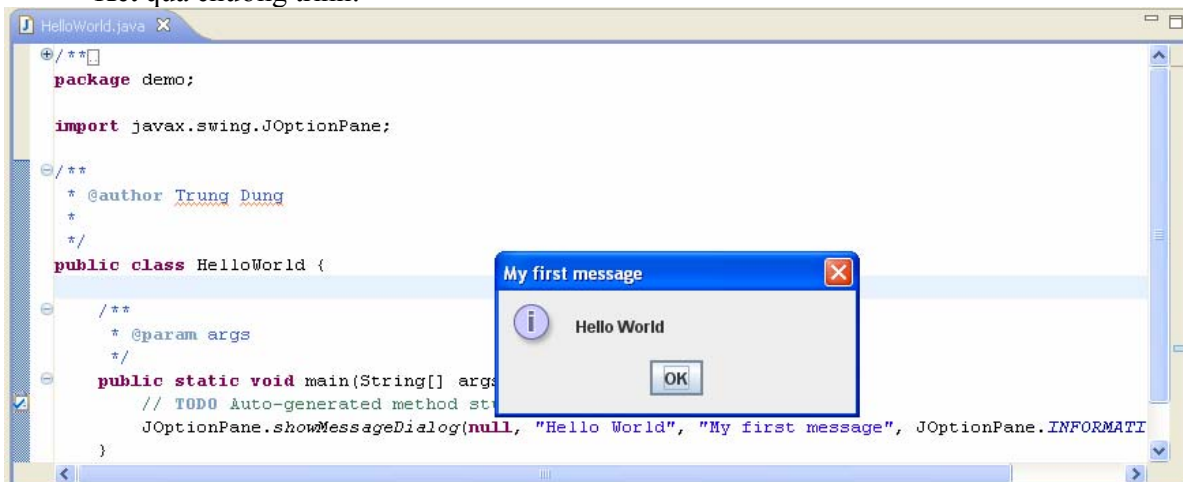
Chọn Run → Run As → Java Application hoặc nhấn Ctrl - F11



Một cửa sổ mới hiện ra hỏi ta có muốn save file này trước khi run không. Hãy check vào ô “Always save...” để sau này nó đừng hỏi nữa và OK.



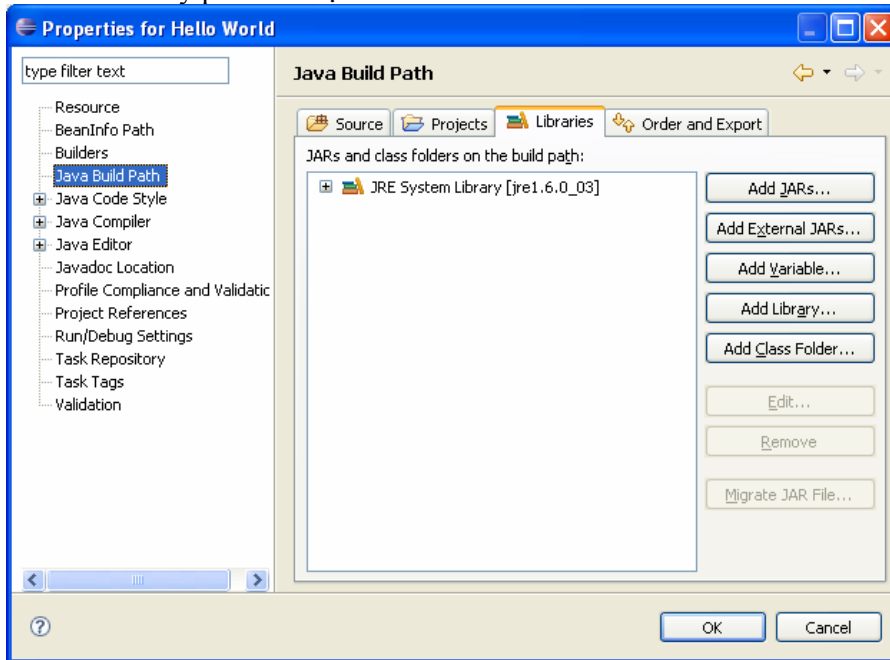
Kết quả chương trình:



4. Cấu hình hệ thống

a. Thêm các thư viện vào project

Các thư viện trong java thường được lưu thành các file jar. Để sử dụng các thư viện này trong project, bạn hãy chọn Project → Properties. Trong cửa sổ mới hiện ra, chọn mục Java Build Path bên tay phải và chọn tab Libraries.



Chọn các mục sau để thêm các file thư viện vào project:

- Add External JARs: Thêm file jar bằng đường dẫn tuyệt đối.
- Add JARs: Thêm file jar hiện đang nằm trong workspace.
- Add Library: Thêm các thư viện có sẵn trong Eclipse, hoặc các thư viện được người dùng cấu hình sẵn trong workspace

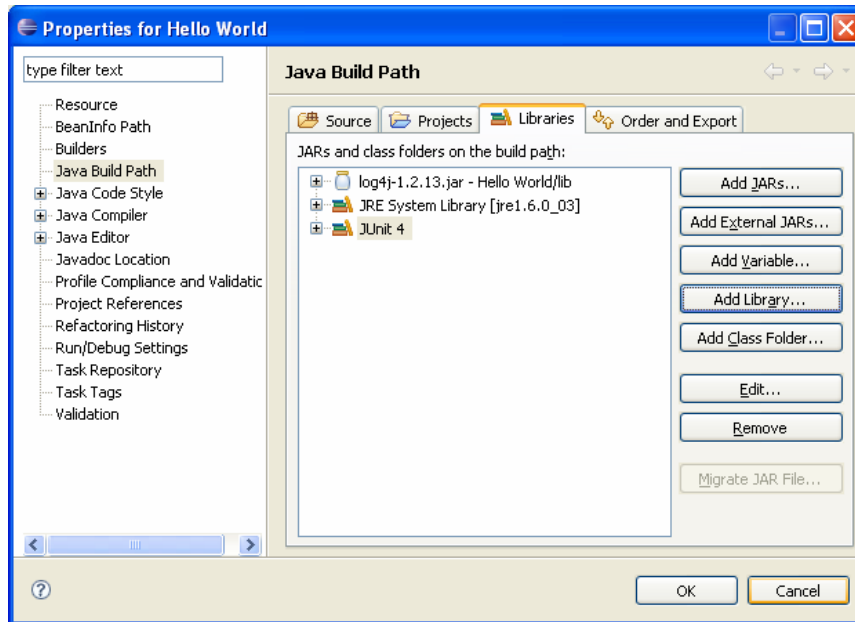
Thông thường, nếu không phải là các thư viện thường dùng có sẵn như J2EE, JUnit, bạn nên copy file thư viện vào project và chọn Add JARs. Eclipse sẽ lưu đường dẫn tương đối đến thư viện đó, và như vậy khi cần có thể copy toàn bộ project qua nơi khác mà không cần cấu hình lại thư viện. Ở đây ta sẽ sử dụng 2 thư viện là Log4j và JUnit

Tạo một thư mục tên lib bên trong project. Có thể tạo bằng cách click phải vào project và chọn New→Folder. Copy file log4j-1.2.13.jar (có thể tìm trên mạng) vào thư mục lib. Sau đó chọn project và nhấn F5 để Eclipse cập nhật lại nội dung thư mục. Mở lại cửa sổ quản lý library của project, chọn Add JARs, chọn file vừa copy vào và OK



Thư viện log4j là một thư viện mã nguồn mở, chuyên dùng cho việc ghi lại (logging) các kết quả xử lý trong chương trình, được dùng rất nhiều trong việc debug.

Tiếp đến ta sẽ thêm thư viện JUnit. Thư viện JUnit cũng là mã nguồn mở, chuyên dùng cho việc thiết kế và chạy các test case. Thư viện này đã có sẵn trong Eclipse nên bạn chỉ cần chọn Add Library → JUnit. Chọn JUnit 4 và Finish. Màn hình quản lý project bây giờ sẽ giống như sau:



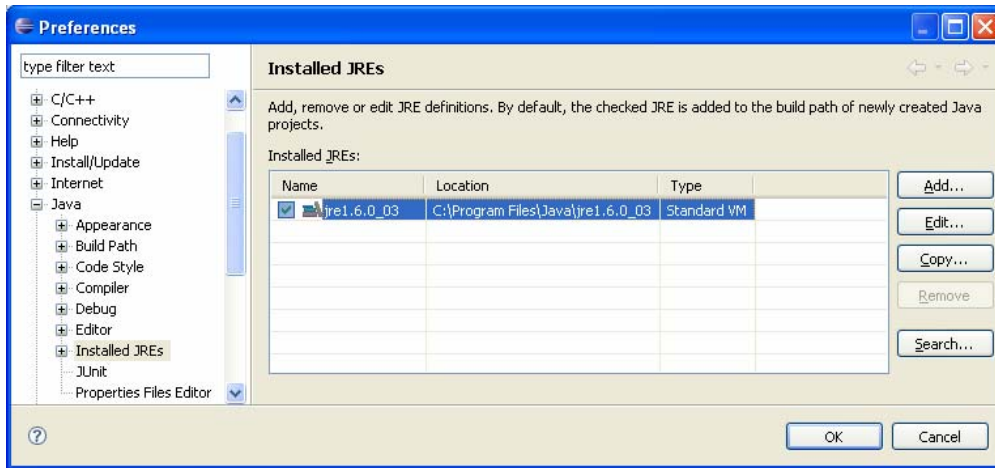
b. Cấu hình jdk và javadoc

Javadoc là bộ tài liệu giống như MSDN của Microsoft, trong đó có mô tả đầy đủ về các package, lớp có sẵn trong java cũng như các phương thức của lớp đó. Chuẩn javadoc là một chuẩn phổ biến mà bất cứ phần mềm mã nguồn mở nào viết bằng java cũng đều đưa ra các tài liệu dạng này. Khi bạn đề con trỏ tại một method và nhấn F2, nội dung trợ giúp được hiển thị lên chính là được lấy trong javadoc

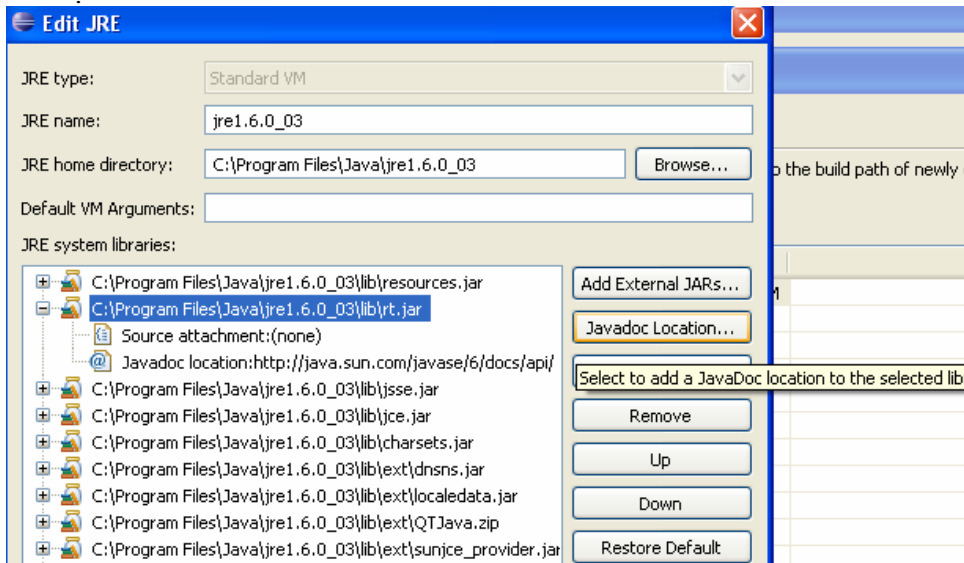


Theo mặc định, Eclipse cấu hình javadoc được lấy ở trang web của Sun. Điều này có thể gây khó khăn nếu không có Internet hoặc đường truyền tốc độ chậm. Ta có thể download toàn bộ javadoc về máy và cấu hình lại đường dẫn này. Javadoc có thể được download ở địa chỉ http://java.sun.com/javase/downloads/index_jdk5.jsp#docs

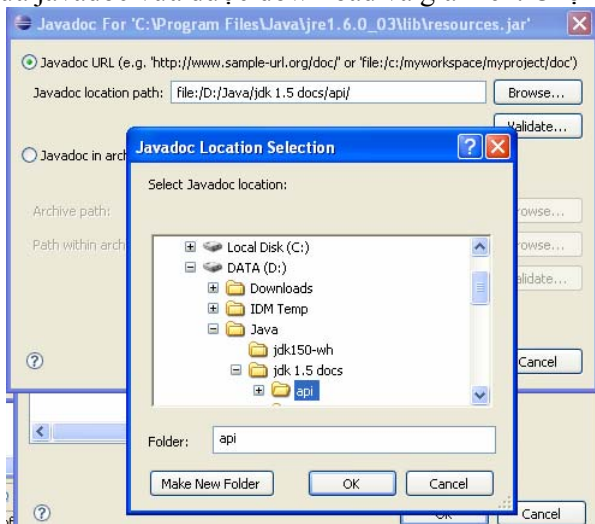
Sau khi download và giải nén, bạn hãy chọn Windows → Preferences. Ở menu bên phải, chọn Java → Installed JREs, chọn phiên bản java tương ứng bên tay phải



Chọn nút Edit. Trong cửa sổ mới hiện ra, chọn thư viện rt.jar trong số các thư viện được liệt kê và chọn Javadoc Location



Cửa sổ mới hiện ra, ta sẽ thiết lập lại đường dẫn javadoc là thư mục api nằm trong thư mục chứa javadoc vừa được download và giải nén. Chọn OK



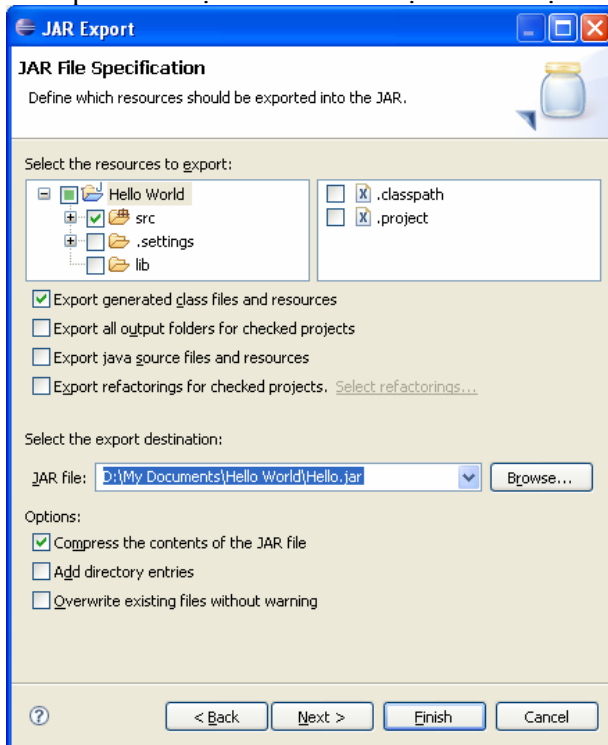
Sau khi thiết lập xong javadoc, bạn sẽ thấy tốc độ hiển thị tooltip nhanh hơn nhiều. Bây giờ, giả sử ta muốn tìm hiểu kỹ thêm về lớp JOptionPane, xem nó có những phương thức nào và có bà con họ hàng gì với những lớp khác. Bạn hãy để con trỏ vào bất cứ chỗ nào có chữ

JOptionPane trên Editor và nhấn Shift-F2. Javadoc của lớp này sẽ được hiển thị trên trình duyệt mặc định của máy bạn.

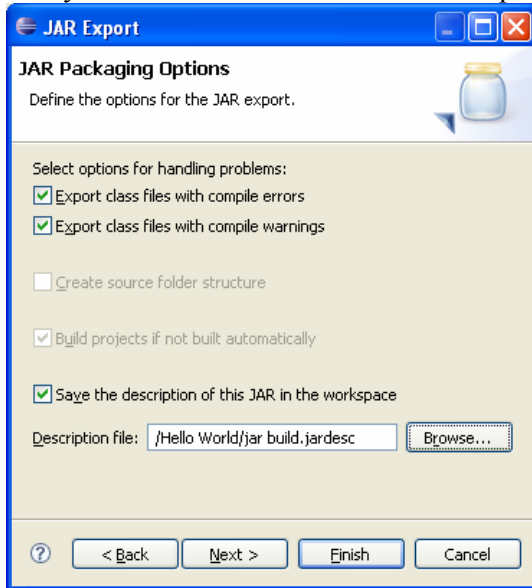


5. Tạo file jar

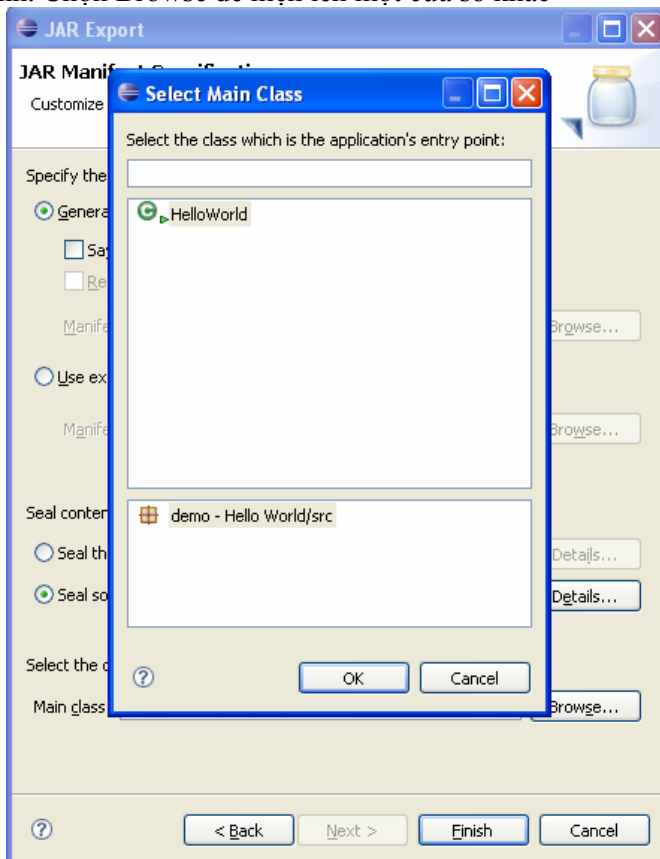
Bây giờ chúng ta sẽ build chương trình thành một file jar để có thể đem qua bất kỳ máy nào có máy ảo Java để chạy, kể cả máy đó là Solaris hay Linux. Bạn hãy chọn File → Export. Trong cửa sổ mới hiện ra, chọn Java → JAR file và click Next. Trong chương trình này, chúng ta chỉ export thư mục src mà thôi. Đặt tên và chọn đường dẫn cho file jar. Chọn Next



Cửa sổ tiếp theo, bạn có thể lưu lại cấu hình của file jar để lần sau không phải lặp lại quá trình này nữa. Check vào ô “Save the description...” và chọn tên file. Xong Next



Màn hình tiếp theo yêu cầu bạn phải config file manifest. Đây là một file miêu tả, cho biết nhiều thông tin về file jar như version, điểm bắt đầu của chương trình và các thư viện sử dụng. Hiện tại, bạn chỉ cần chú ý ô textbox ở cuối cùng, nơi bạn sẽ chỉ ra điểm nhập của chương trình. Chọn Browse để hiện lên một cửa sổ khác



Chương trình của chúng ta chỉ có một điểm nhập duy nhất. Chọn class HelloWorld và OK. Bây giờ bạn sẽ thấy khung Main class sẽ hiện lên dòng chữ demo.HelloWorld. Finish để hoàn tất. Bây giờ hãy dùng cửa sổ Package explorer và refresh project (F5) để kiểm tra 2 file jar và

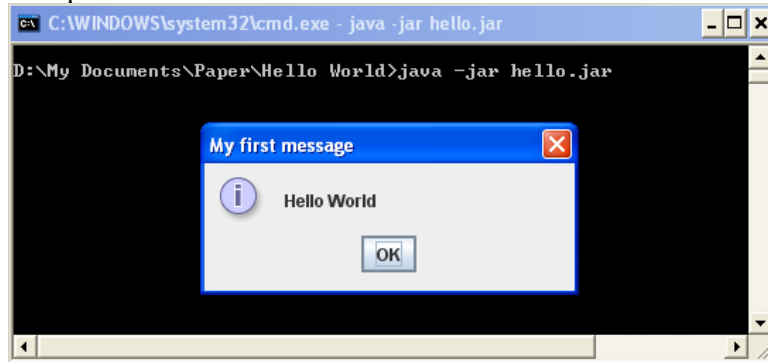
jardesc mới tạo ra. Với file jardesc lưu cấu hình quá trình tạo file jar, lần sau bạn chỉ cần nhấp chuột phải vào nó và chọn Create JAR để tạo lại file jar mới.

Bây giờ chúng ta thử chạy file jar vừa mới tạo. Với Windows, cách đơn giản nhất là double click vào file đó. Tuy nhiên với những môi trường khác, ta nên dùng màn hình console để gọi.

Mở màn hình console và di chuyển đến thư mục chứa file jar vừa mới tạo. Ta sẽ gõ lệnh sau:

`java -jar hello.jar`

Kết quả:



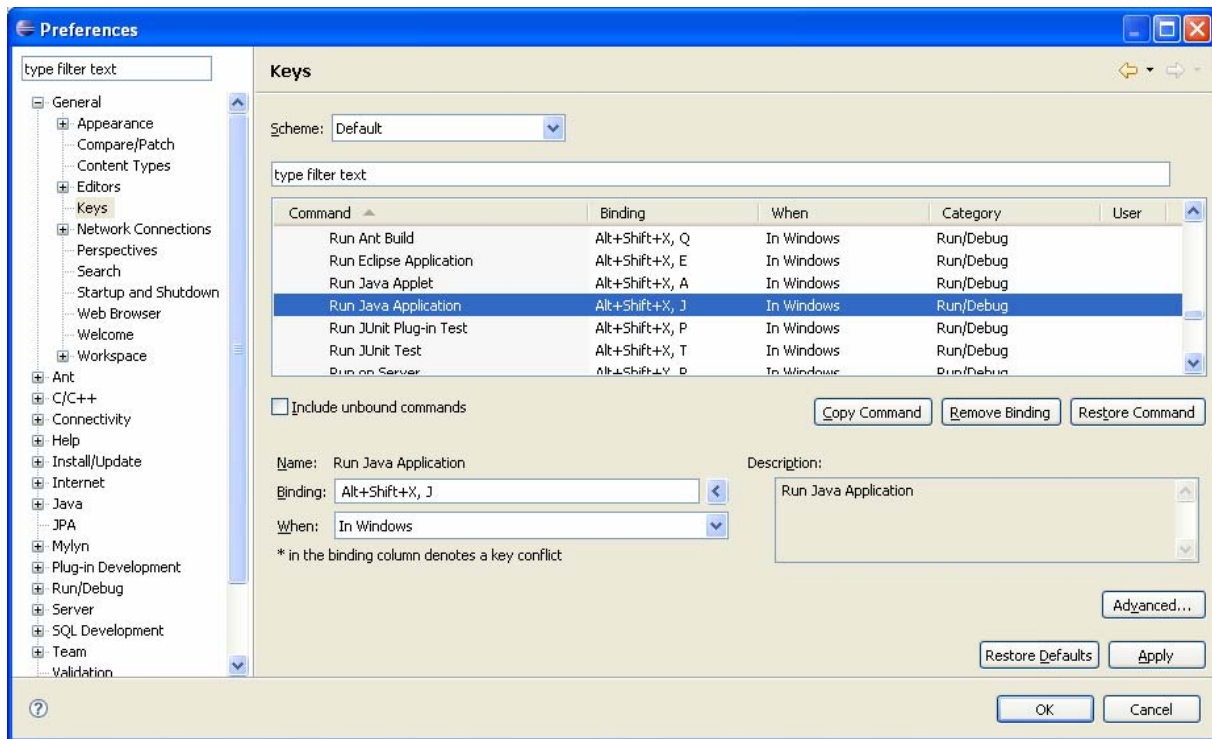
Nếu dòng lệnh trên báo lỗi, rất có thể do máy của bạn chưa thiết lập đường dẫn đến thư mục cài đặt java. Trên thực tế, việc build một project java thường được thực hiện bằng công nghệ Ant (<http://ant.apache.org>, được tích hợp sẵn trong Eclipse) do phải giải quyết các vấn đề về thư viện và cấu hình. Cách tạo file jar ở trên chỉ là một cách đơn giản, không thích hợp với các project lớn. Bạn nên tìm hiểu công nghệ Ant để biết cách build một project java hoàn chỉnh.

6. Các phím tắt trong Eclipse

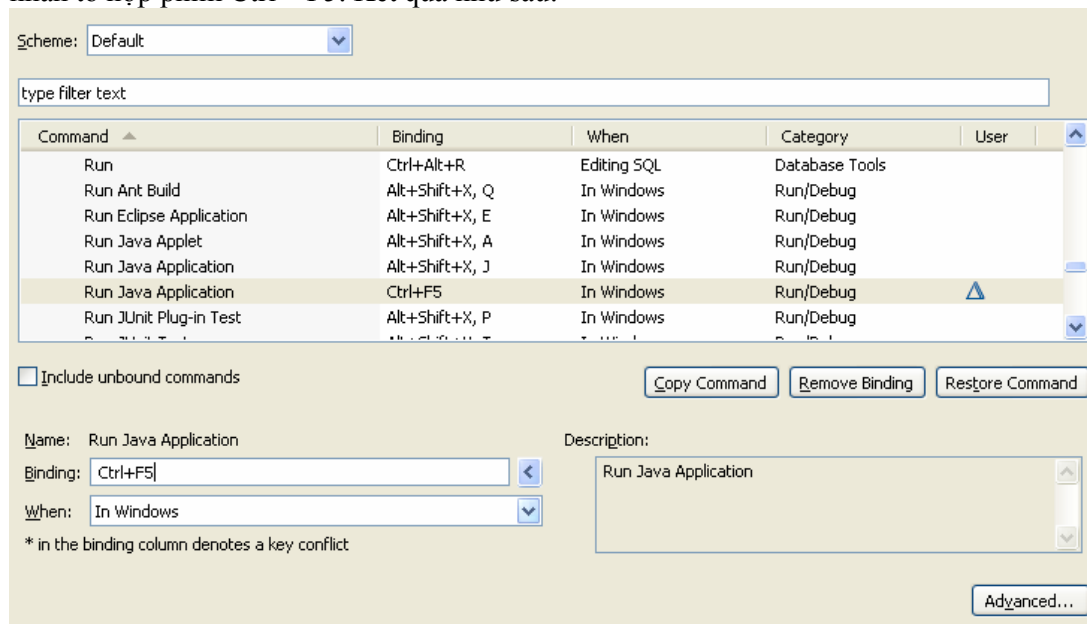
Để xem các phím tắt thông dụng trong Eclipse, bạn hãy nhấn tổ hợp phím Ctrl – Shift – L. Sau đây là một số chức năng thường sử dụng nhất

- comment/uncomment 1 dòng: Ctrl - /
- comment 1 block (dùng ký hiệu /* và */): Ctrl – Shift - /
- uncomment 1 block: để con trỏ ở bất kỳ vị trí nào trong block và nhấn Ctrl – Shift - \
- tự động import các class: Ctrl – Shift – O
- tự động thêm các lệnh try/catch: Alt – Shift – Z
- xem javadoc tại vị trí con trỏ: F2
- xem javadoc tại cửa sổ trình duyệt: Shift – F2
- hỗ trợ theo ngữ cảnh (hiện list các phương thức, tự động điền tên biến, ...): Ctrl – Space
- xóa một dòng: Ctrl – D
- sửa lỗi nhanh: đưa con trỏ tới vị trí bị lỗi và Ctrl – 1 (số một)
- tự động giống hàng cho cả file: Ctrl – A và sau đó Ctrl – I

Hệ thống các phím tắt của Eclipse khá mạnh, có thể hỗ trợ việc viết code rất nhanh. Tuy nhiên, một số phím tắt quá dài và khó nhớ. Ví dụ tổ hợp phím để “Run Java application” là Alt-Shift-X, J. Ta sẽ thiết lập lại tổ hợp này thành tổ hợp quen thuộc Ctrl – F5. Bạn hãy chọn Windows → Preferences. Chọn mục General → Keys, chọn Run Java Application bên tay phải

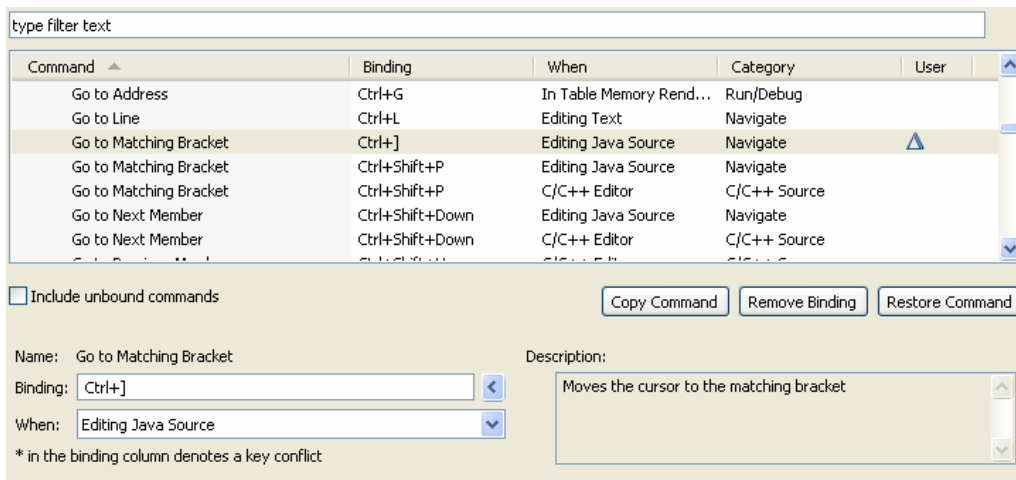


Bạn hãy nhấn vào nút Copy command để tạo ra một lệnh mới. Khi con trỏ đang ở ô Binding, nhấn tổ hợp phím Ctrl – F5. Kết quả như sau:

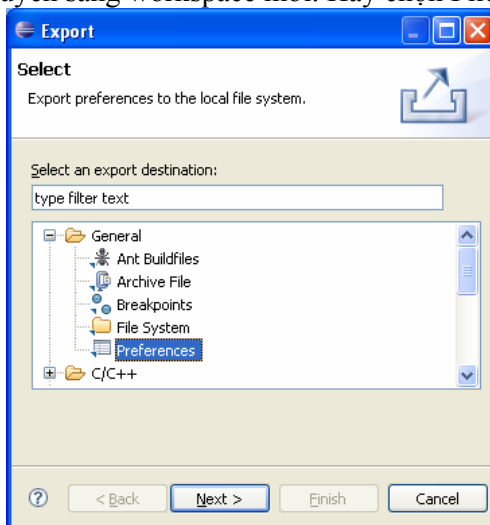


OK để thoát ra. Giờ hãy thử bằng cách nhấn Ctrl – F5 để chạy chương trình của bạn.

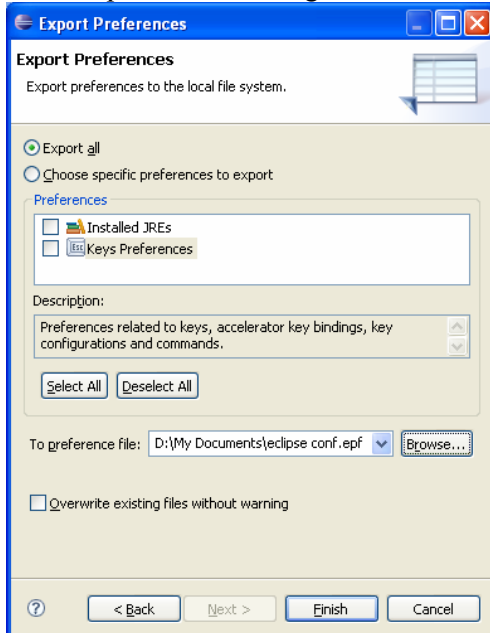
Tương tự như vậy, khi viết code với nhiều cặp dấu ngoặc lồng nhau, bạn có thể muốn xem các cặp dấu ngoặc tương ứng với nhau thế nào. Hãy áp dụng cách ở trên để thêm vào phím tắt cho chức năng Go to Matching Bracket là Ctrl -], một phím tắt phổ biến trong các editor khác.



Sau khi đã thiết lập một số lượng khá khá các phím tắt, bạn có thể muốn lưu lại cấu hình này để sau này sử dụng khi qua workspace khác. Lưu ý là các thiết lập trên sẽ đều bị mất khi bạn chuyển sang workspace mới. Hãy chọn File → Export, chọn mục General → Preferences.



Chọn export all hoặc từng loại cấu hình mà bạn muốn lưu lại, đặt tên file và Finish.



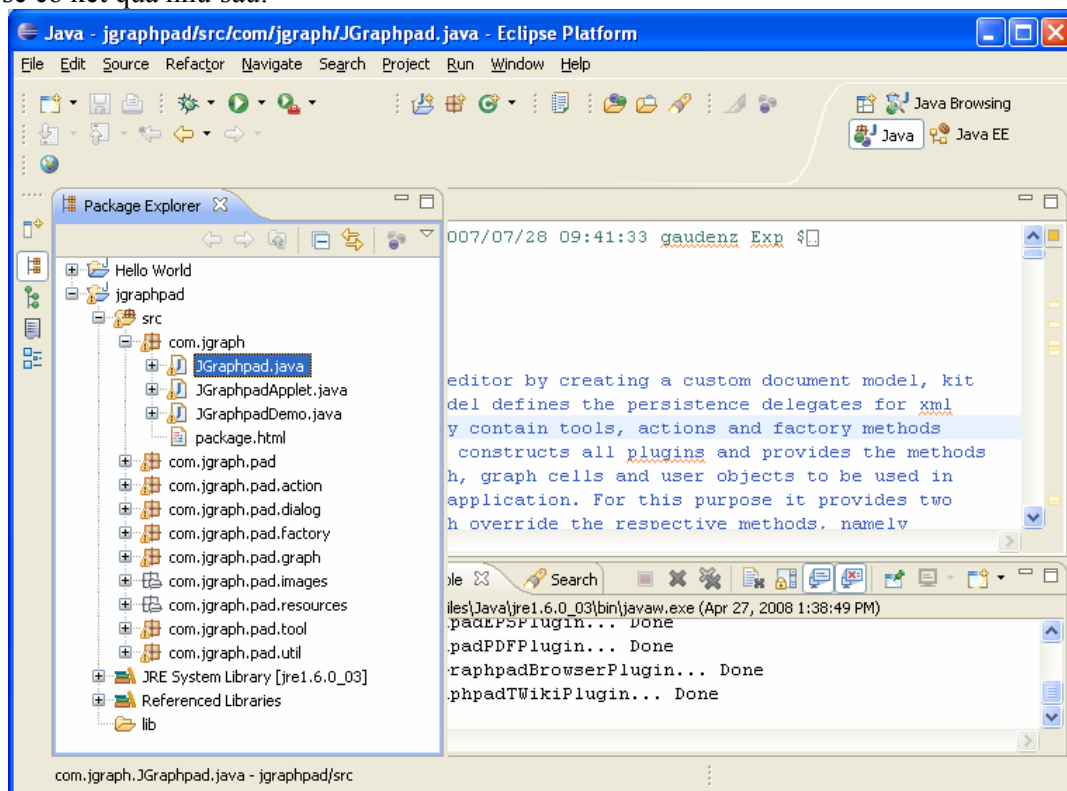
Sau này khi chuyển sang workspace mới, bạn chỉ cần chọn File → Import, mục General → Preferences.

7. Import một project có sẵn vào trong Eclipse

Nếu project có sẵn được tạo bằng Eclipse hoặc một tool phổ biến khác như JBuilder, bạn chỉ cần vào File → Import. Chọn mục General → Existing Projects into Workspace.

Nếu project cũ không được tạo bằng Eclipse, bạn phải tạo một project mới và copy source code của project cũ vào trong project mới.

Chúng ta hãy thử import một project phức tạp hơn vào trong workspace, và kiểm tra các tính năng khác của Eclipse. Bạn hãy chép file jgraphpad.rar vào thư mục hiện chứa workspace và giải nén. Đây là file chứa một project đã được tạo sẵn. Nếu bạn không có file này, bạn hãy download mã nguồn của thư viện jgraph ở địa chỉ <http://www.jgraph.com/> và làm theo cách thứ 2 ở trên. Nếu bạn đã có file này, hãy làm theo cách thứ nhất để import project. Lúc này, chúng ta sẽ có kết quả như sau:



JGraph là một thư viện mã nguồn mở viết bằng java, chuyên hỗ trợ việc vẽ và xử lý các đối tượng đồ họa. Lúc này ta đã có một project tương đối phức tạp để vọc các tính năng khác của Eclipse.

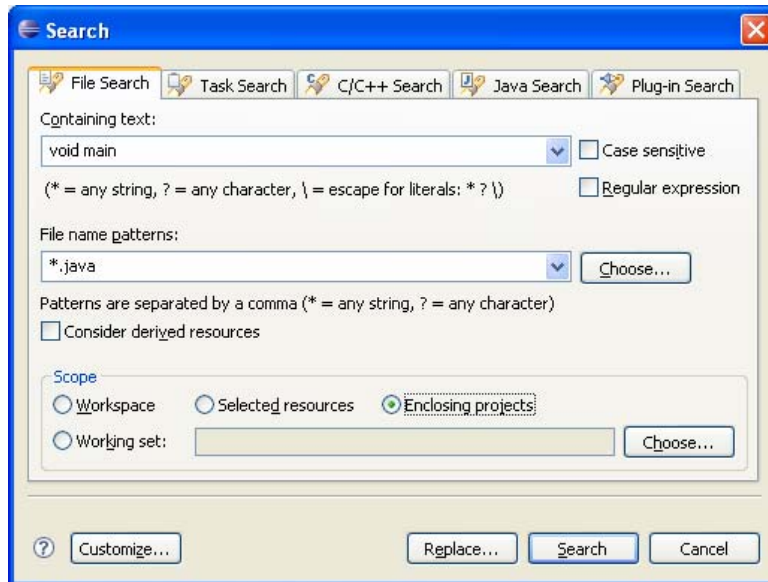
8. Các tiện ích trong Eclipse

a. Chức năng tìm kiếm

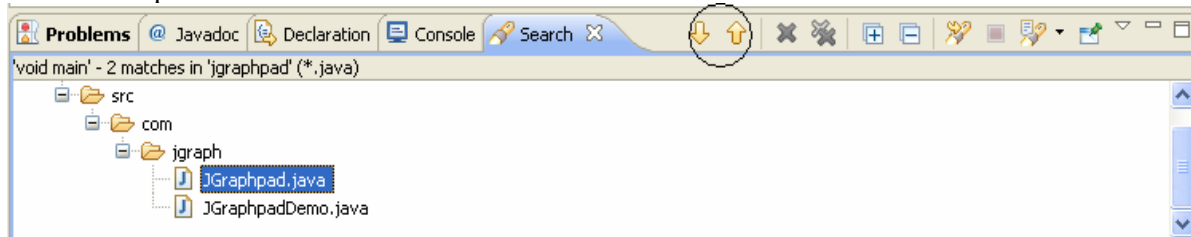
Để tìm kiếm một từ hoặc cụm từ trong editor hiện hành: Ctrl – F

Để tìm kiếm một cụm từ trong toàn bộ project, chọn Search → File...

Ta thử áp dụng điều này để tìm điểm nhập (hàm main) của chương trình. Chọn Search → File. Một cửa sổ mới sẽ hiện ra. Trong ô Containing text gõ vào chuỗi “void main”. Chọn File name patterns là *.java, như vậy ta sẽ chỉ tìm trong các file mã nguồn mà bỏ qua tất cả các loại file config khác. Trong khung scope, nếu chọn Workspace, Eclipse sẽ tìm trong tất cả các project đang mở; nếu chọn Enclosing projects, Eclipse sẽ chỉ tìm kiếm trong project chứa file đang được edit. Sau khi thiết lập các thông số, hãy nhấn Search



Kết quả tìm kiếm:

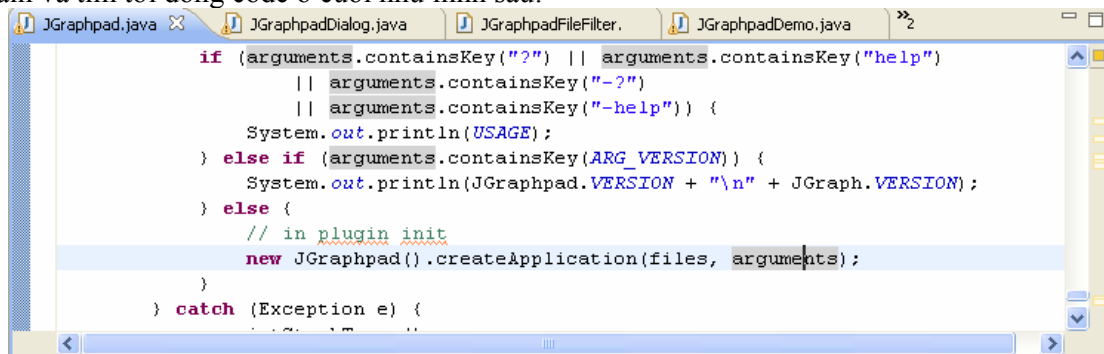


Bạn hãy sử dụng các button có hình dạng mũi tên để duyệt kết quả tìm kiếm. Hình trên cho thấy có 2 class chứa hàm main, trong đó lớp JGraphpad chính là điểm nhập của chương trình. Hãy mở file đó và chạy thử chương trình.

Ngoài ra, Eclipse còn hỗ trợ nhiều chế độ tìm kiếm khác như tìm các phương thức, gói hoặc lớp trong chức năng search tổng quát Search → Search...

b. Các tiện ích trace code

Bây giờ chúng ta hãy vọc sơ qua hàm main vừa tìm được một chút. Bạn hãy xem qua hàm main và tìm tới dòng code ở cuối như hình sau:

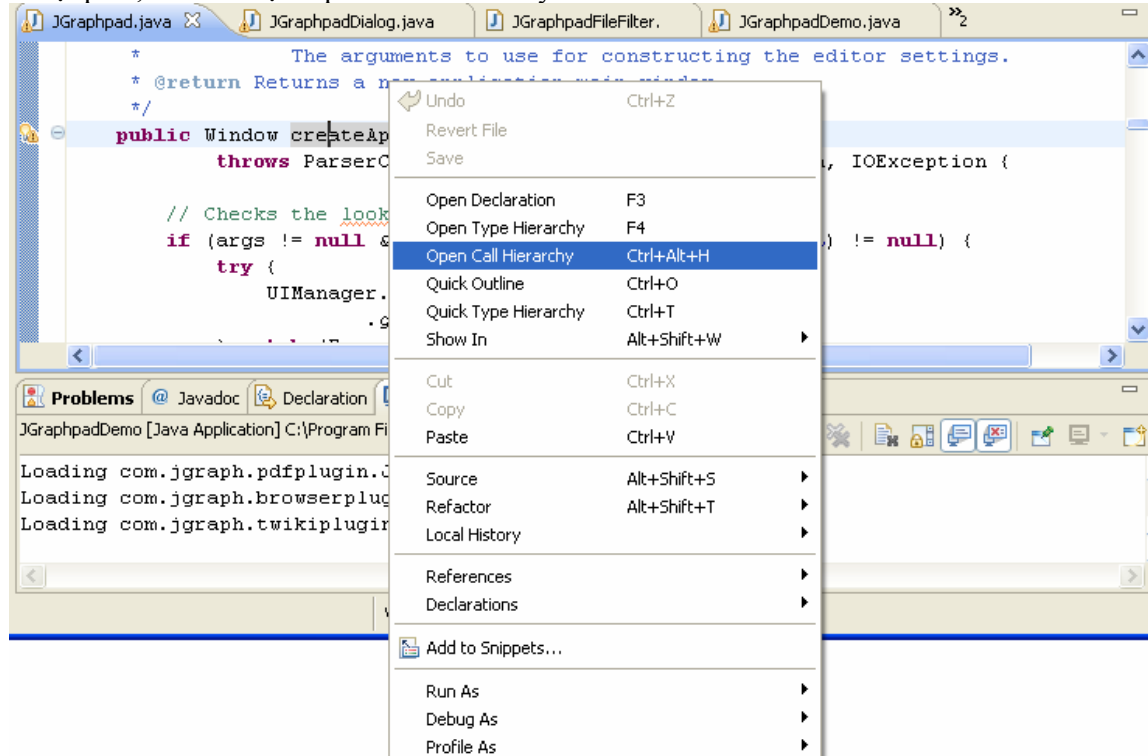


Hãy để ý là khi bạn để con trỏ tại biến arguments đủ lâu, tất cả các xuất hiện của biến này đều được tô màu. Đây là một tiện ích rất mạnh của Eclipse, nó sẽ giúp bạn đọc code dễ dàng hơn.

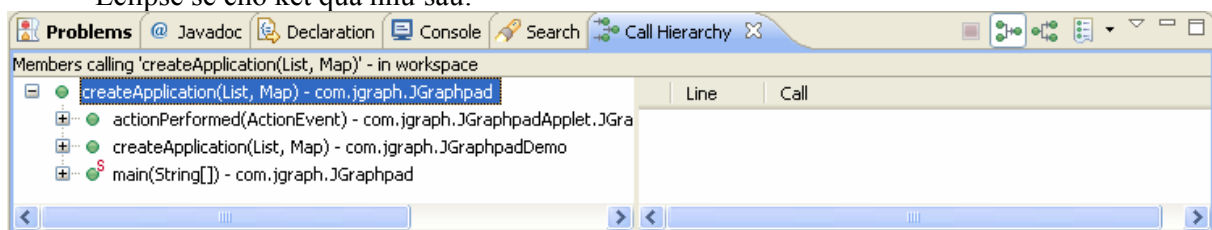
Bây giờ hãy xem dòng lệnh bên dưới chú thích “//in plugin init”, đây chắc hẳn là đoạn code chính để khởi tạo giao diện. Có thể bạn muốn đọc tiếp xem cái hàm *createApplication* này làm gì. Rất đơn giản, hãy giữ nút Ctrl và nhấp chuột vào phương thức *createApplication*, Eclipse sẽ tự động mở một file mới và nhảy đến đúng vị trí định nghĩa đoạn code này cho bạn. Không chỉ

tìm đến định nghĩa của các phương thức, bạn cũng có thể giữ Ctrl và click vào một biến để nhảy đến định nghĩa của biến đó.

Đôi khi bạn có một phương thức và muốn biết nó được gọi bởi những phương thức nào. Ta hãy thử tìm xem ngoài hàm main vừa tìm được ở trên, có phương thức nào cũng gọi hàm *createApplication* hay không. Hãy để con trỏ chuột bên trong chuỗi *createApplication* và nhấn chuột phải, sau đó chọn Open Call Hierarchy



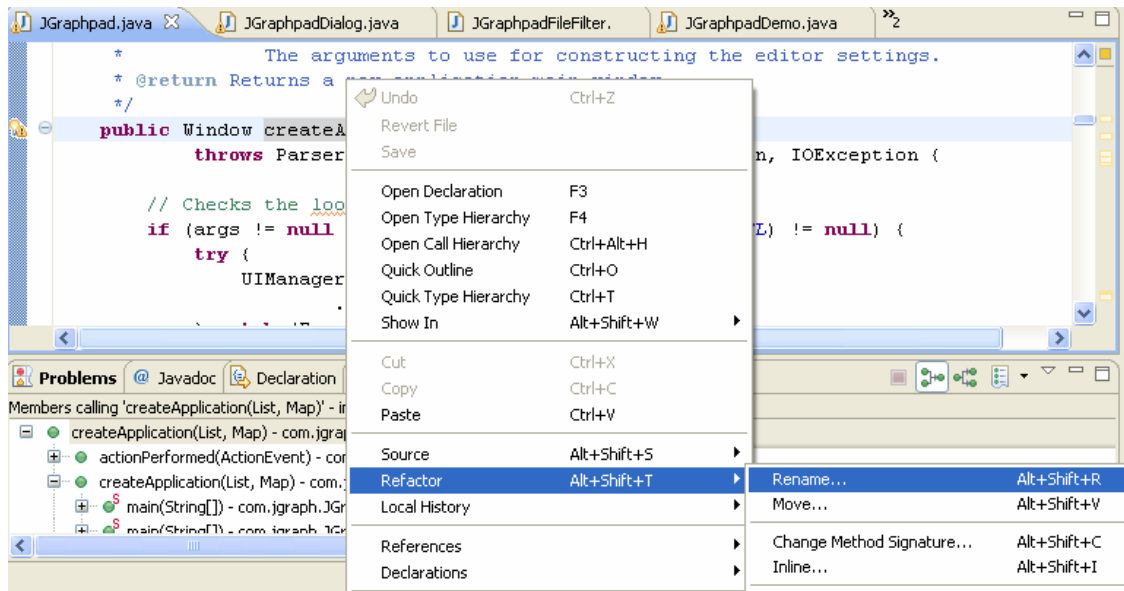
Eclipse sẽ cho kết quả như sau:



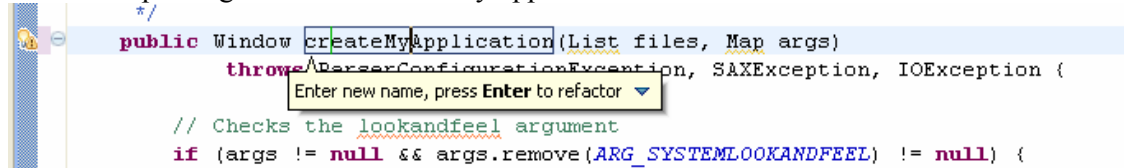
Hãy sử dụng các dấu + để tiếp tục theo vết (trace) các phương thức. Đôi lúc, bạn sẽ thấy chức năng này rất hữu hiệu.

c. Chức năng Refactor

Một trong những chức năng rất mạnh của Eclipse là khả năng giúp người sử dụng thay đổi code một cách đồng loạt và ít rủi ro nhất. Hãy thử thay đổi tên của phương thức *createApplication* thành *createMyApplication*. Bạn có thể hình dung mức độ khó khăn của tác vụ này: tất cả các đoạn code gọi hàm này đều phải được sửa lại. Bây giờ, hãy để con trỏ vào bất cứ vị trí xuất hiện nào của phương thức *createApplication*, click chuột phải, chọn Refactor → Rename



Sửa tên phương thức thành createMyApplication và Enter để kết thúc



Bây giờ hãy chú ý đến hàm main ban đầu, Eclipse đã tự động sửa tên phương thức cho chúng ta.

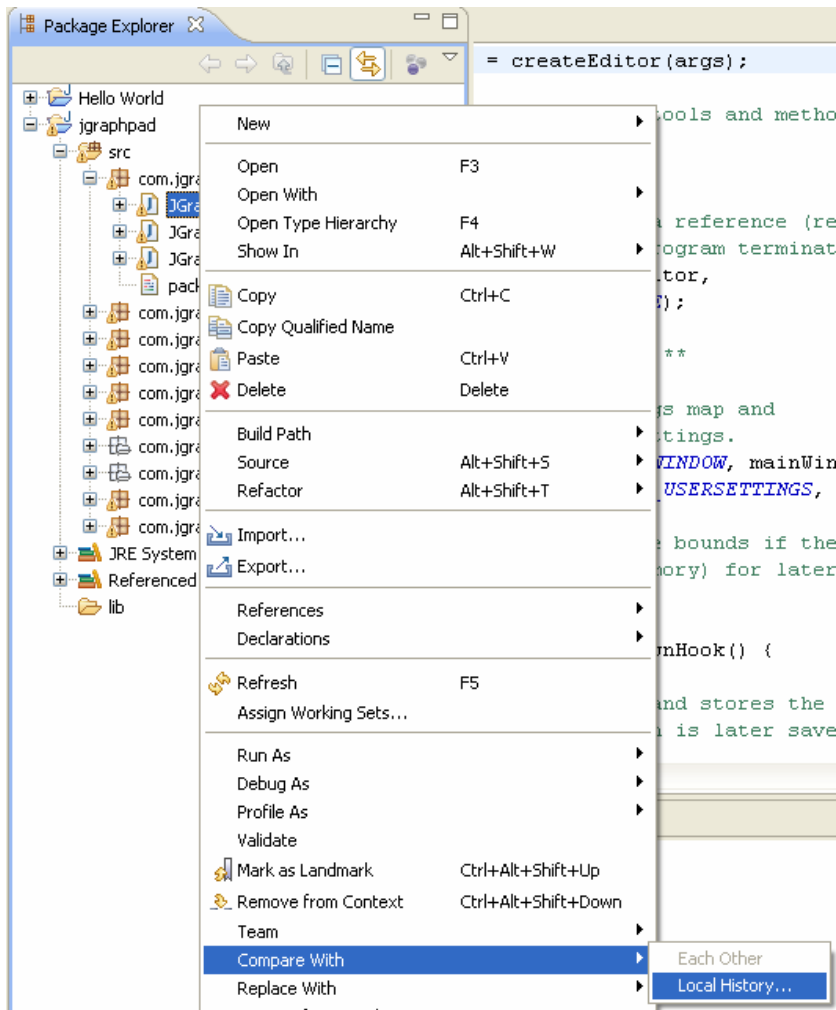


Chức năng Refactor còn có thể làm nhiều tác vụ khác: đổi tên class và package, di chuyển một class từ package này sang các package khác (đồng nghĩa với các lệnh import phải thay đổi theo), hay thậm chí là thay đổi signature của một phương thức.

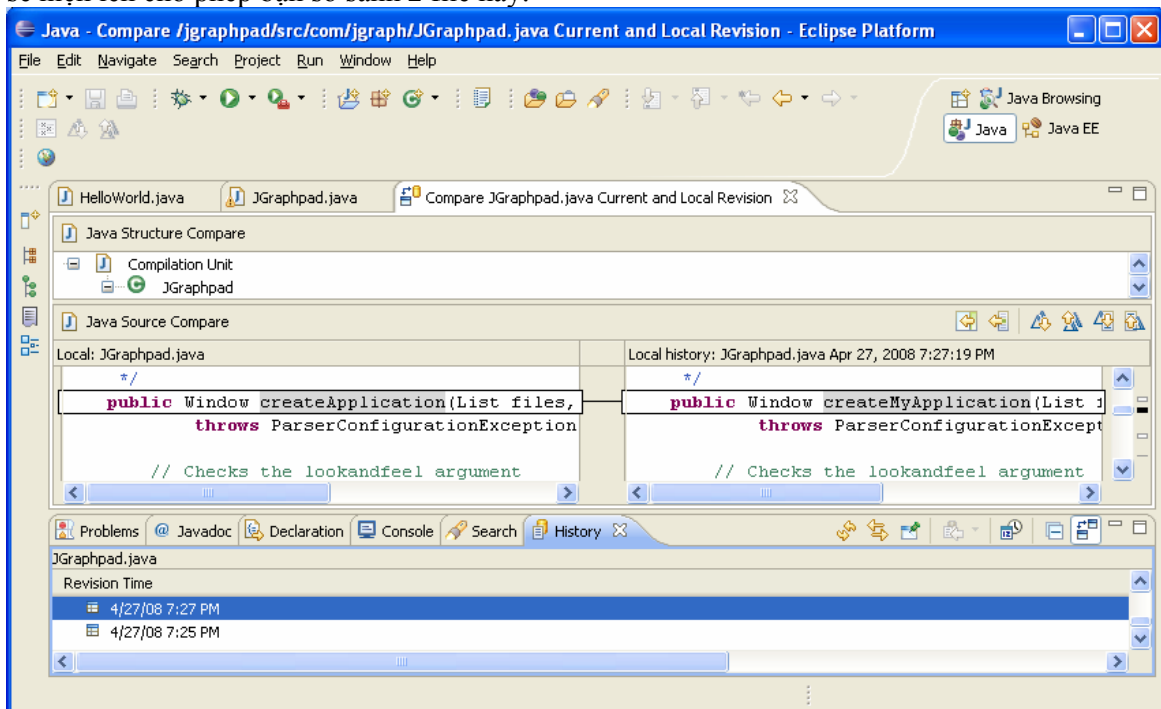
Ngoài ra Eclipse còn hỗ trợ các tính năng như: tự động sinh các hàm get và set cho một biến, tự động sinh các constructor. Bạn có thể thử các chức năng này bằng cách click chuột phải và chọn Source...

9. Khôi phục lại các file cũ

Sau một hồi vọc đủ các tính năng refactor, có thể bạn sẽ “hối hận” chẳng? Thật may, Eclipse có thể giúp bạn xem lại các thay đổi cũng như khôi phục các file nguồn trở về trạng thái trước đó. Để xem lại các thay đổi, bạn hãy click chuột phải vào file và chọn Compare with → Local history



Trong khung History, bạn hãy double click vào phiên bản mà Eclipse đã lưu lại. Một cửa sổ sẽ hiện lên cho phép bạn so sánh 2 file này.

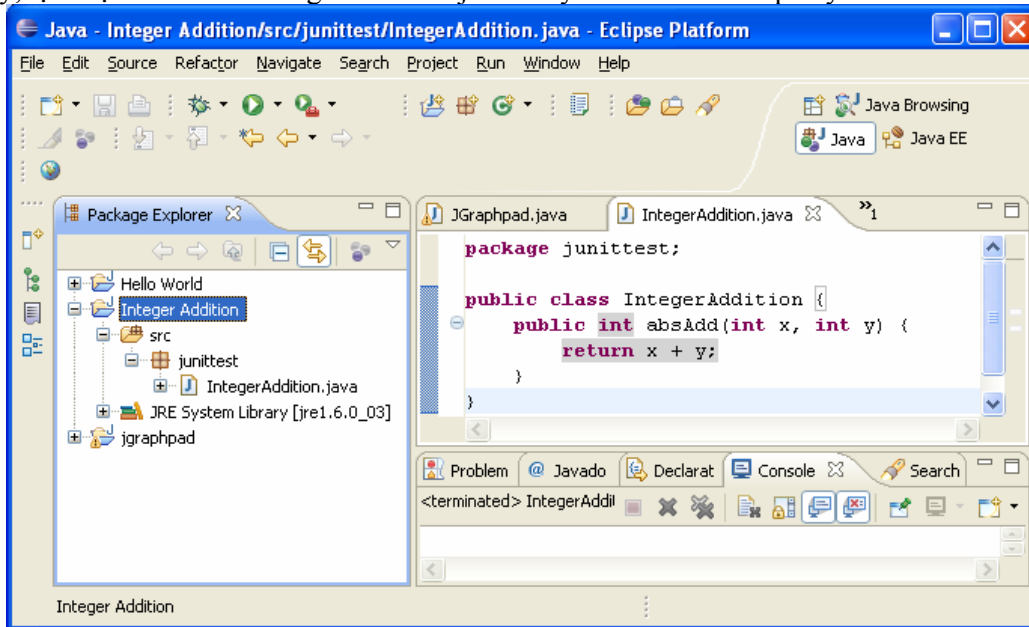


Như bạn thấy, Eclipse sẽ highlight các thay đổi trong file, giúp ta có thể track lại dễ dàng các thay đổi trong chương trình. Để phục hồi lại một file nào đó, cách làm cũng tương tự, nhưng thay vì chọn mục Compare with thì bạn hãy chọn Replace with → Local history.

10. Chức năng testing với JUnit

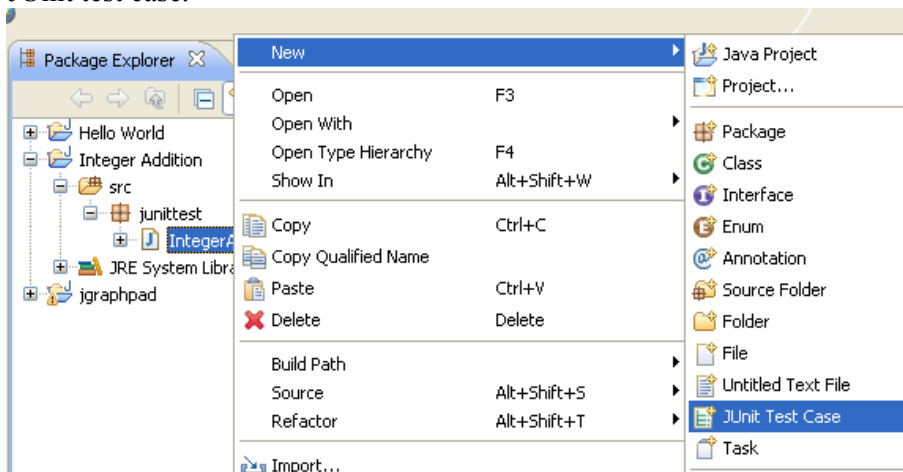
JUnit là một gói phần mềm mã nguồn mở đi kèm với Eclipse, chuyên dùng cho việc thiết kế và chạy thử các test case. Ví dụ này sẽ cho bạn một số khái niệm về JUnit. Bạn có thể tìm hiểu thêm về JUnit tại trang web www.junit.org

Chúng ta hãy tạo một project mới trong Eclipse, lấy tên là Integer Addition. Trong project này, tạo một file mới là IntegerAddition.java. Hãy viết code cho lớp này như hình sau:

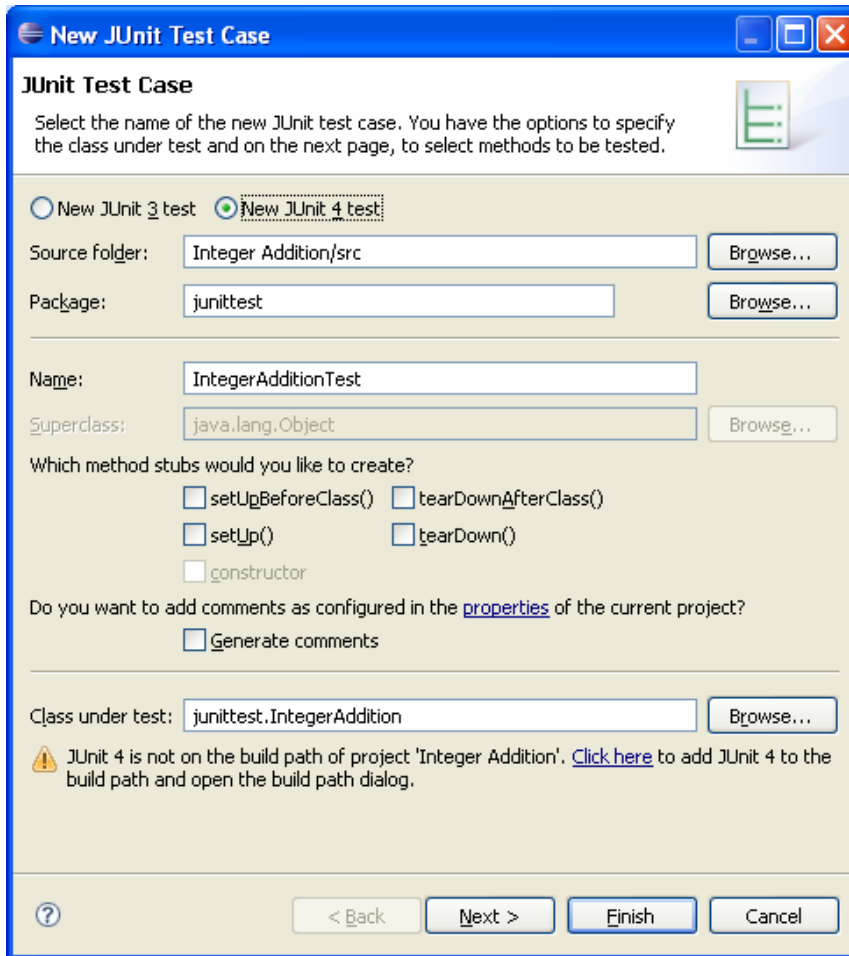


Trong lớp này, ta có một phương thức *absAdd*, mục đích của nó là cộng *giá trị tuyệt đối* của 2 số nguyên. Như bạn thấy, chúng ta đã cố tình hiện thực sai phương thức này để thử nghiệm tính năng của JUnit.

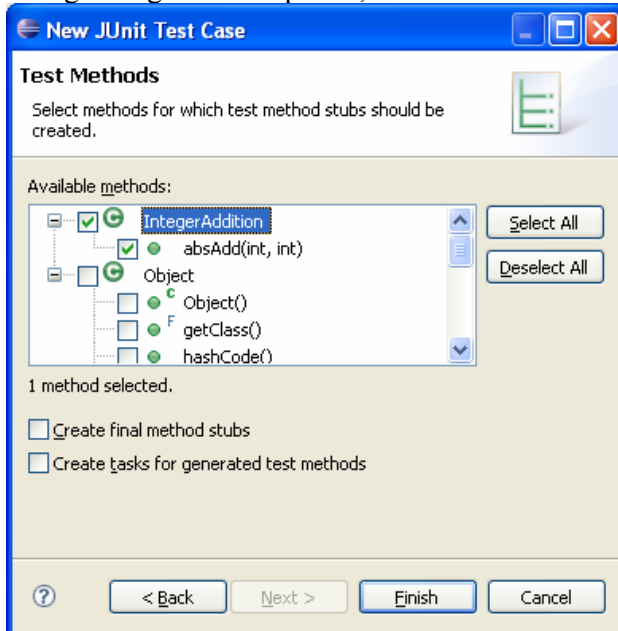
Hãy tạo một JUnit test case bằng cách click chuột phải vào file IntegerAddition, chọn New → JUnit test case.



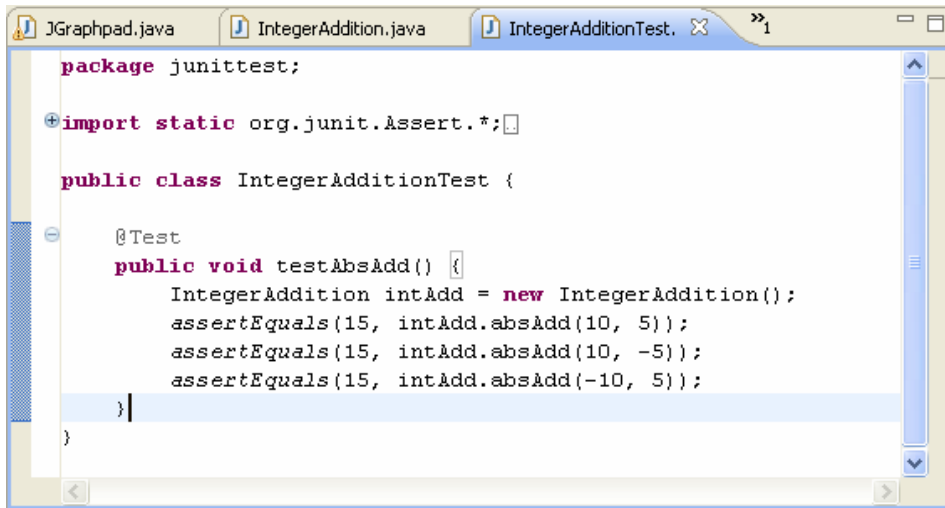
Cửa sổ mới hiện ra, hãy chọn New JUnit 4 test. Dấu cảnh báo ở dưới cho biết thư viện JUnit chưa được include vào project của bạn. Hãy chọn Click here và OK để Eclipse tự động thêm vào. Sau đó chọn Next.



Trong khung cửa sổ tiếp theo, check vào ô `absAdd` như hình dưới:



Sau đó OK, bạn sẽ có một lớp mới tên là `IntegerAdditionTest`. Lớp này đã có sẵn phương thức `testAbsAdd`. Bạn có thể viết thêm các phương thức khác, nhưng chỉ những phương thức có chỉ thị `@Test` ở đầu mới được JUnit test thử. Bạn hãy hiện thức các phương thức này như sau:



```
package junittest;

import static org.junit.Assert.*;

public class IntegerAdditionTest {

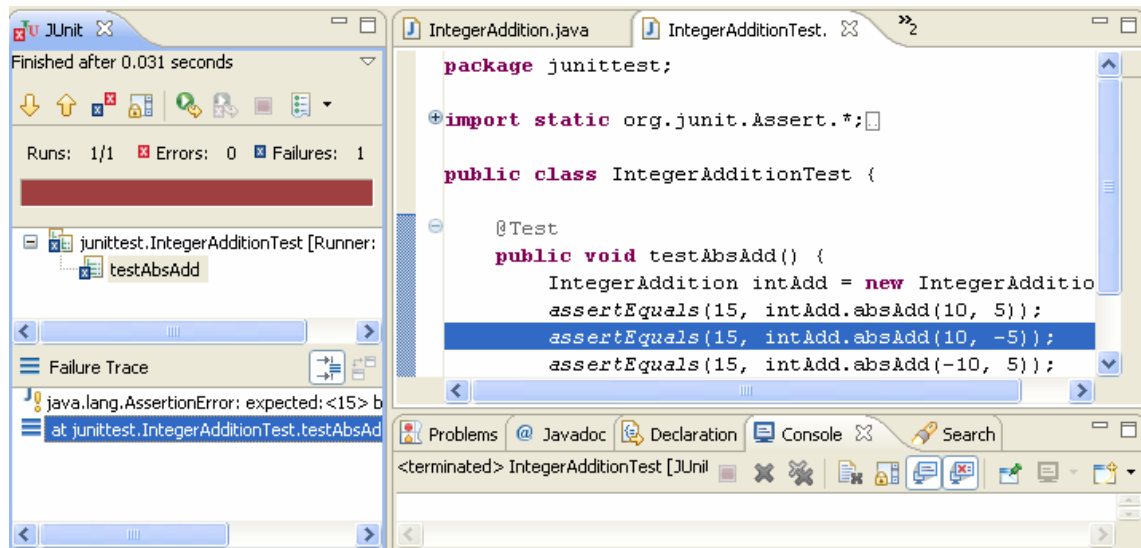
    @Test
    public void testAbsAdd() {
        IntegerAddition intAdd = new IntegerAddition();
        assertEquals(15, intAdd.absAdd(10, 5));
        assertEquals(15, intAdd.absAdd(10, -5));
        assertEquals(15, intAdd.absAdd(-10, 5));
    }
}
```

Ý nghĩa của đoạn code trên là như sau: đầu tiên ta khai báo một biến *intAdd* thuộc lớp *IntegerAddition* và gọi constructor của lớp này. Tiếp theo là 3 hàm *assertEquals* dùng để test cho 3 trường hợp.

Các hàm *assertEquals* cho thấy các “kỳ vọng” về kết quả đạt được của chương trình. Ý nghĩa của 3 dòng test trên là: ta hy vọng hàm *absAdd* sẽ trả về giá trị 15 khi nhận được input là 10 và 5; và tương tự, hy vọng hàm này sẽ trả về giá trị 15 khi nhận các cặp input lần lượt là 10 và -5, -10 và 5.

Mỗi khi gặp hàm *assertEquals*, JUnit sẽ kiểm tra xem hai thông số được đưa vào có bằng nhau không. Nếu bằng, chương trình sẽ chạy tiếp; nếu không, JUnit sẽ dừng phương thức được test lại và báo về một failure. Một phương thức bị failure sẽ không ảnh hưởng gì đến các phương thức cần test khác.

Bây giờ hãy chạy thử test case này. Hãy chọn Run → Run As → JUnit Test. Kết quả như sau:



```
package junittest;

import static org.junit.Assert.*;

public class IntegerAdditionTest {

    @Test
    public void testAbsAdd() {
        IntegerAddition intAdd = new IntegerAddition();
        assertEquals(15, intAdd.absAdd(10, 5));
        assertEquals(15, intAdd.absAdd(10, -5));
        assertEquals(15, intAdd.absAdd(-10, 5));
    }
}
```

JUnit cho ta biết có 1 test case bị hỏng, đồng thời nếu bạn double click vào dòng trong khung Failure Trace ở dưới, nó sẽ cho bạn biết chương trình bị fail ở test case nào. Bây giờ hãy thử hiện thực lại lớp *IntegerAdditionTest* theo hình dưới đây



```
package junittest;

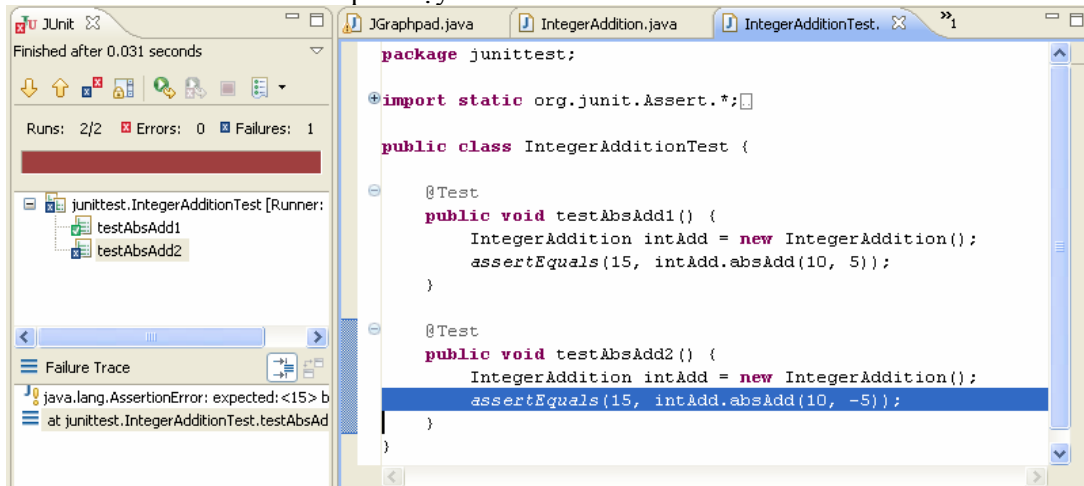
import static org.junit.Assert.*;

public class IntegerAdditionTest {

    @Test
    public void testAbsAdd1() {
        IntegerAddition intAdd = new IntegerAddition();
        assertEquals(15, intAdd.absAdd(10, 5));
    }

    @Test
    public void testAbsAdd2() {
        IntegerAddition intAdd = new IntegerAddition();
        assertEquals(15, intAdd.absAdd(10, -5));
    }
}
```

Ta đã có 2 test case. Kết quả chạy thử như sau:



```
package junittest;

import static org.junit.Assert.*;

public class IntegerAdditionTest {

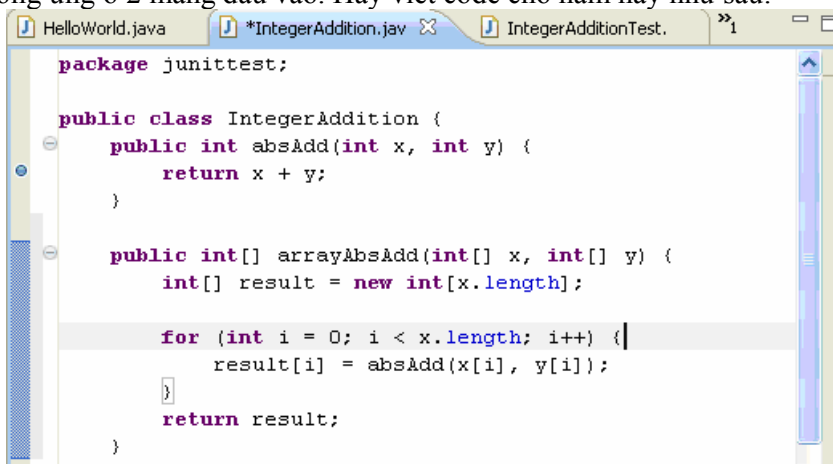
    @Test
    public void testAbsAdd1() {
        IntegerAddition intAdd = new IntegerAddition();
        assertEquals(15, intAdd.absAdd(10, 5));
    }

    @Test
    public void testAbsAdd2() {
        IntegerAddition intAdd = new IntegerAddition();
        assertEquals(15, intAdd.absAdd(10, -5));
    }
}
```

Như bạn thấy, JUnit thông báo đã chạy được 2 test case và có một failure.

11. Tính năng debug

JUnit chỉ có thể giúp cho bạn biết chương trình chạy đúng hay sai. Để biết chương trình chạy sai chỗ nào, bạn cần có các kỹ năng debug. Chúng ta hãy thử mở rộng chương trình vừa viết bằng cách thêm một hàm tên là *arrayAbsAdd*. Hàm này sẽ nhận vào input là 2 mảng số nguyên và output là một mảng số nguyên có các phần tử là tổng trị tuyệt đối của các phần tử tương ứng ở 2 mảng đầu vào. Hãy viết code cho hàm này như sau:



```
package junittest;

public class IntegerAddition {

    public int absAdd(int x, int y) {
        return x + y;
    }

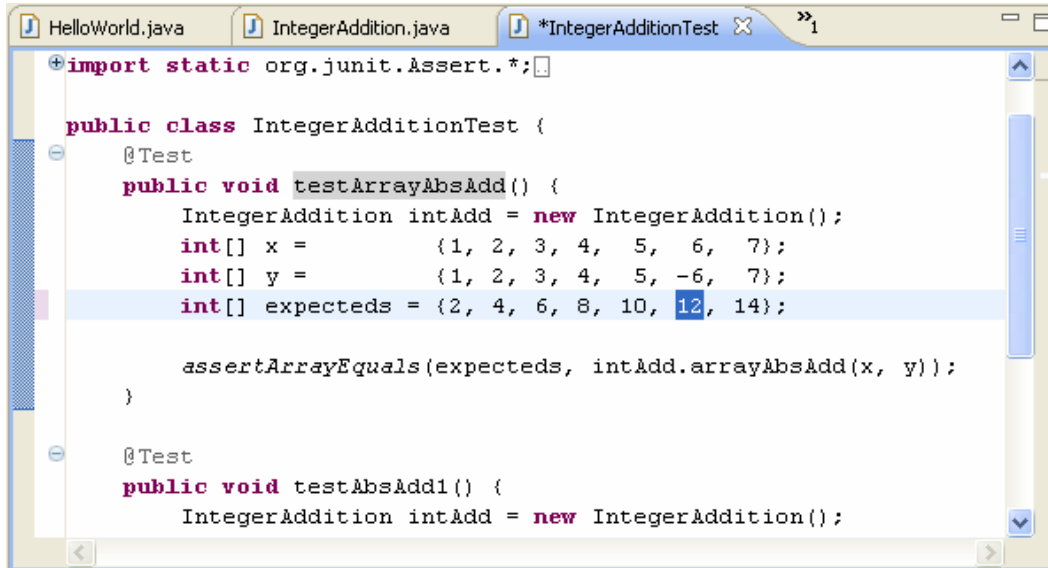
    public int[] arrayAbsAdd(int[] x, int[] y) {
        int[] result = new int[x.length];

        for (int i = 0; i < x.length; i++) {
            result[i] = absAdd(x[i], y[i]);
        }

        return result;
    }
}
```


Đoạn code trong hàm trên khởi tạo một mảng `result` có chiều dài bằng chiều dài của mảng `x`. Để đơn giản, chúng ta giả sử rằng 2 mảng đầu vào đều khác null và có chiều dài bằng nhau. Vòng lặp `for` sẽ lần lượt gán các giá trị cho mảng `result` bằng cách gọi lại hàm `absAdd` ở trên. Như bạn thấy, hàm `absAdd` chúng ta đã hiện thực sai, nên dẫn tới kết quả của `arrayAbsAdd` cũng sẽ không chính xác.

Sau khi hiện thực xong hàm `arrayAbsAdd`, hãy save lại và ta sẽ bắt đầu viết một test case mới như sau:



```
import static org.junit.Assert.*;

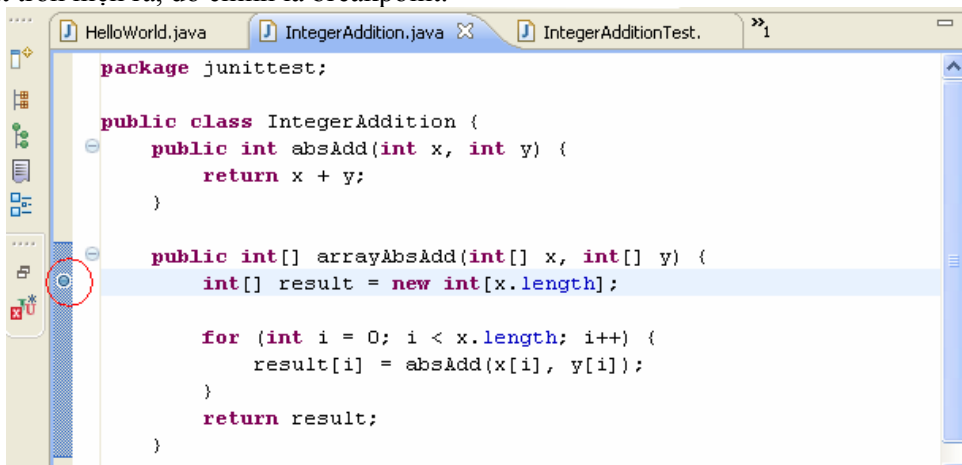
public class IntegerAdditionTest {
    @Test
    public void testArrayAbsAdd() {
        IntegerAddition intAdd = new IntegerAddition();
        int[] x = {1, 2, 3, 4, 5, 6, 7};
        int[] y = {1, 2, 3, 4, 5, -6, 7};
        int[] expecteds = {2, 4, 6, 8, 10, 12, 14};

        assertEquals(expecteds, intAdd.arrayAbsAdd(x, y));
    }

    @Test
    public void testAbsAdd1() {
        IntegerAddition intAdd = new IntegerAddition();
```

Bạn hãy chú ý giá trị 12 trong mảng `expecteds`, đó chính là chỗ sai của chương trình. Còn bây giờ hãy chạy thử JUnit để kiểm tra, JUnit sẽ báo bạn sai 2/3 test case (kể cả test case vẫn còn ở phần trước). Chúng ta hãy thử debug phương thức `arrayAbsAdd` trước, mặc dù trên thực tế, bạn phải luôn debug bắt đầu từ phương thức đơn giản hơn.

Công việc đầu tiên là thiết lập các breakpoint tại những điểm bạn nghi ngờ có thể sai. Mỗi khi chương trình chạy đến 1 breakpoint, nó sẽ dừng lại cho bạn kiểm tra. Giả sử bạn chưa có ý niệm gì về lỗi sai của chương trình, hãy đặt một breakpoint ngay đầu của hàm `arrayAbsAdd`. Bạn hãy để ý một thanh dọc ở bên trái của editor, mỗi khi double click vào thanh đó sẽ có một nút tròn hiện ra, đó chính là breakpoint.



```
package junittest;

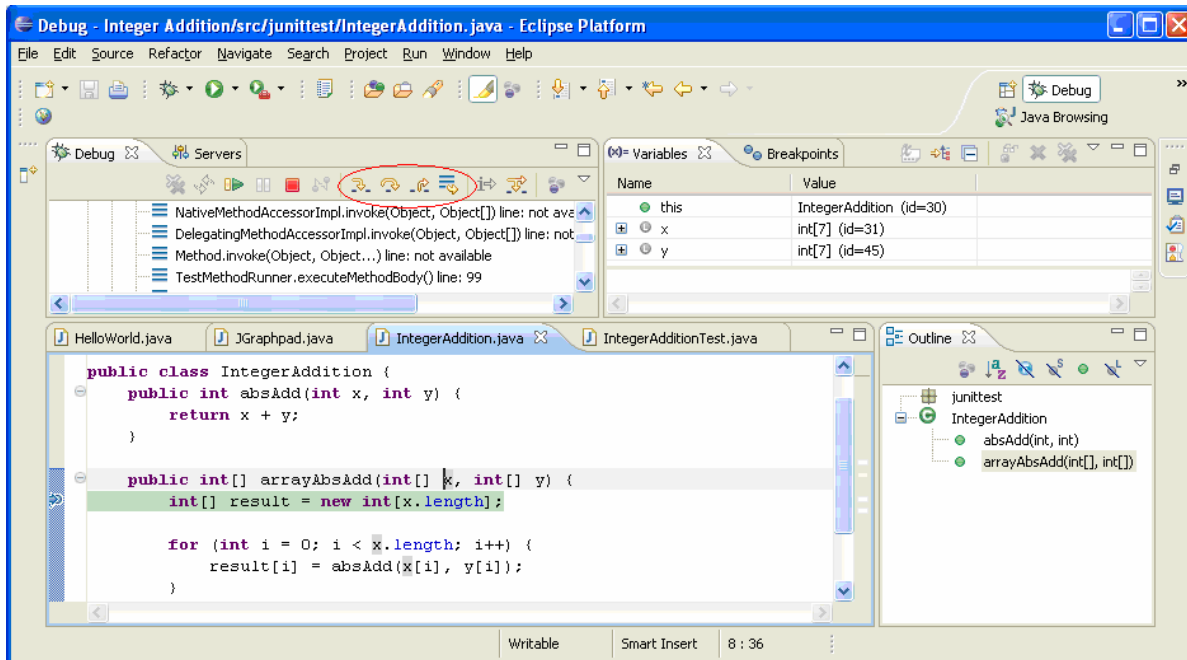
public class IntegerAddition {
    public int absAdd(int x, int y) {
        return x + y;
    }

    public int[] arrayAbsAdd(int[] x, int[] y) {
        int[] result = new int[x.length];

        for (int i = 0; i < x.length; i++) {
            result[i] = absAdd(x[i], y[i]);
        }

        return result;
    }
}
```

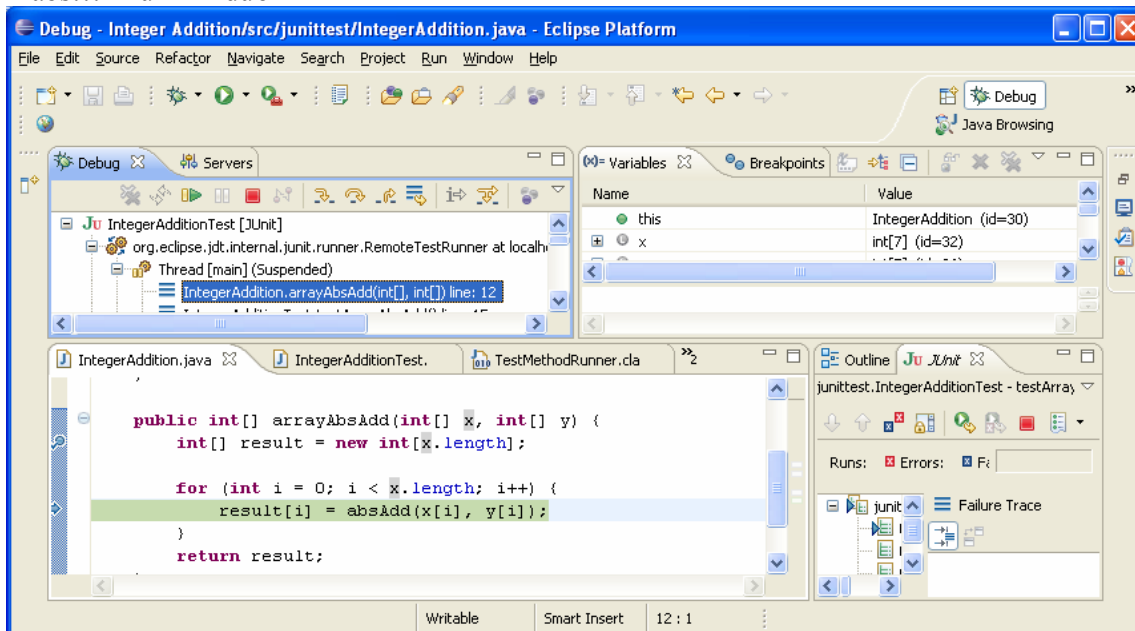
Bây giờ hãy quay lại file `IntegerAdditionTest`. Chọn `Run` → `Debug As` → `JUnit Test`. Một cửa sổ hiện ra hỏi bạn có muốn chuyển qua giao diện Debug không, hãy chọn `Yes`. Đây là giao diện debug của Eclipse.



Như bạn thấy, chương trình dừng lại tại đúng breakpoint mà ta đã thiết lập. Khung Variables cho phép bạn theo dõi các giá trị của các biến đang dùng, còn khung Debug chứa các nút điều khiển việc debug. Các nút điều khiển này gồm:

- _ Step into (F5): nếu gặp một phương thức, lệnh này sẽ chuyển điều khiển vào bên trong phương thức đó
- _ Step over (F6): nếu gặp một phương thức, lệnh này sẽ thực thi phương thức đó và dừng lại sau khi đã thực hiện xong
- _ Step return (F7): trở về phương thức đã gọi phương thức đang debug

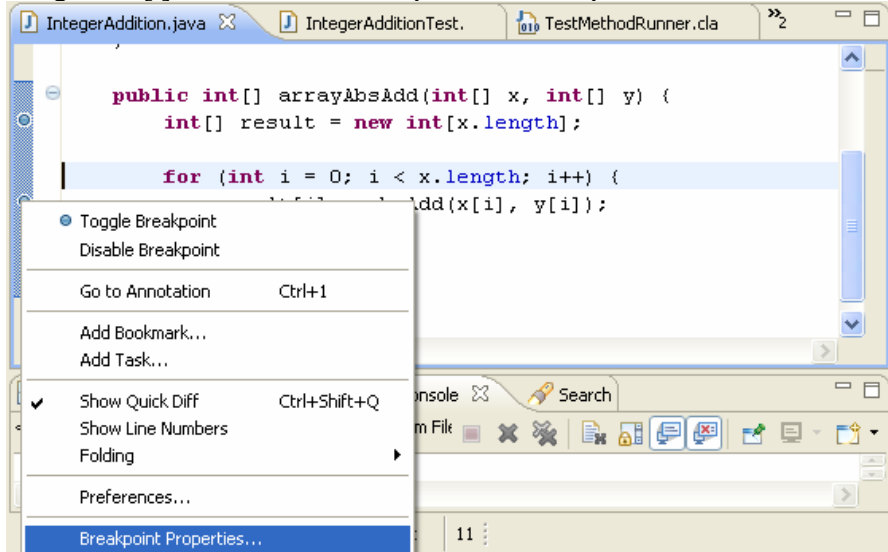
Bạn hãy sử dụng các phím F5 và F6 để điều khiển chương trình chạy tới dòng lệnh `result[i] = abs...` như hình dưới



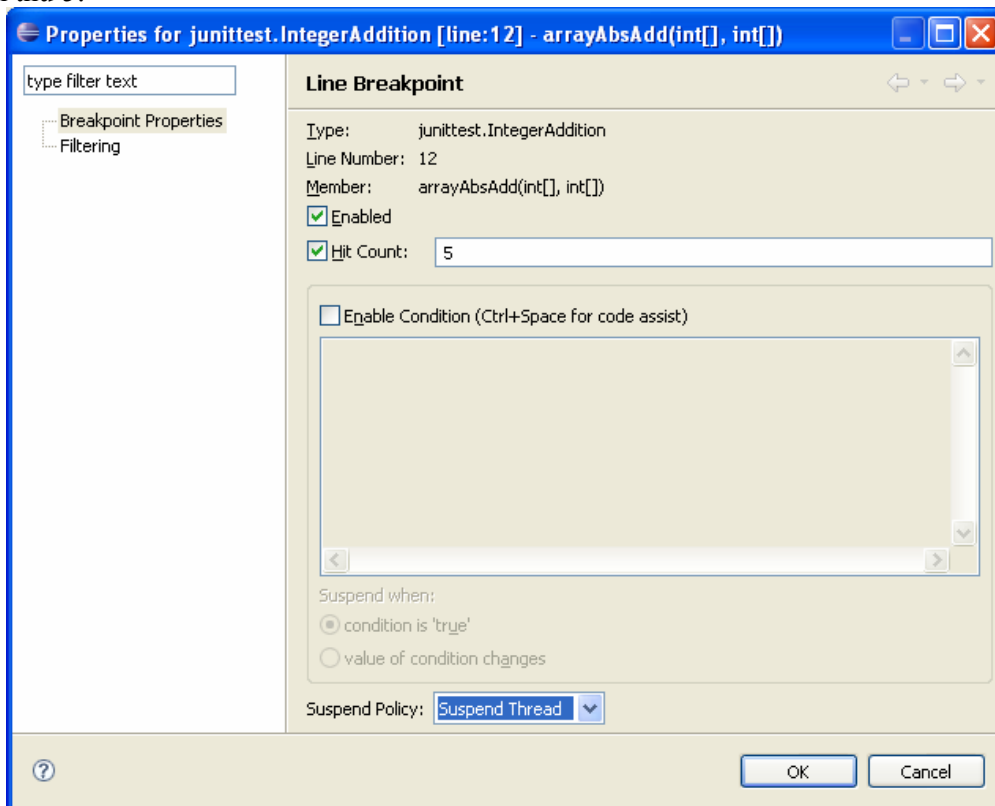
Lúc này, nếu bạn nhấn F5 (step into), chương trình sẽ nhảy đến đoạn code của phương thức `absAdd`; nếu nhấn F6 (step over), chương trình sẽ nhảy qua đến đoạn code của lệnh `for`, tức là đã kết thúc một lần lặp. Bạn hãy tiếp tục trace theo chương trình, đồng thời theo dõi biến `result`

trong khung Variables để kiểm tra. Bạn sẽ dễ dàng phát hiện được chỗ sai của chương trình ở lần lặp for thứ 6 (tương ứng giá trị $i = 5$).

Ví dụ trên rất đơn giản. Nhưng bây giờ hãy giả sử rằng bạn có một vòng for lặp 1000 lần, và lỗi sai được nghi ngờ ở lần lặp thứ ... 999, không lẽ ta phải bấm F6 999 lần để tới được chỗ cần kiểm tra? Rất may là bạn không phải làm vậy. Hãy trở về cửa sổ editor và đặt một breakpoint tại dòng `result[i] = absAdd...`. Click phải vào breakpoint vừa tạo và chọn Breakpoint Properties



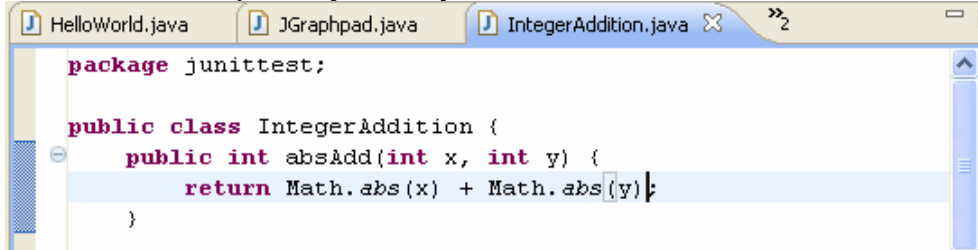
Một cửa sổ mới hiện ra cho phép bạn thiết lập các điều kiện để dừng tại một breakpoint. Giá trị trong ô Hit Count bằng 5 có nghĩa là chương trình chỉ dừng lại khi đã gặp breakpoint này tới lần thứ 5.



Bạn cũng có thể thiết lập các điều kiện dừng phức tạp hơn trong khung Enable Condition (ví dụ $i \geq 5$ hay $x[i] == 4$ chẳng hạn). Hãy OK và test thử để xem hiệu quả của việc thiết lập các breakpoint này.

Để xóa tất cả các breakpoint, bạn chọn Run → Remove All Breakpoints. Bạn nên làm điều này trước khi tạo một file jar, vì các breakpoint sẽ được đính kèm theo file jar của bạn. Điều này dẫn đến việc nếu ai đó sử dụng file jar của bạn, khi debug chương trình, họ sẽ bị breakpoint trong file jar của bạn chặn lại.

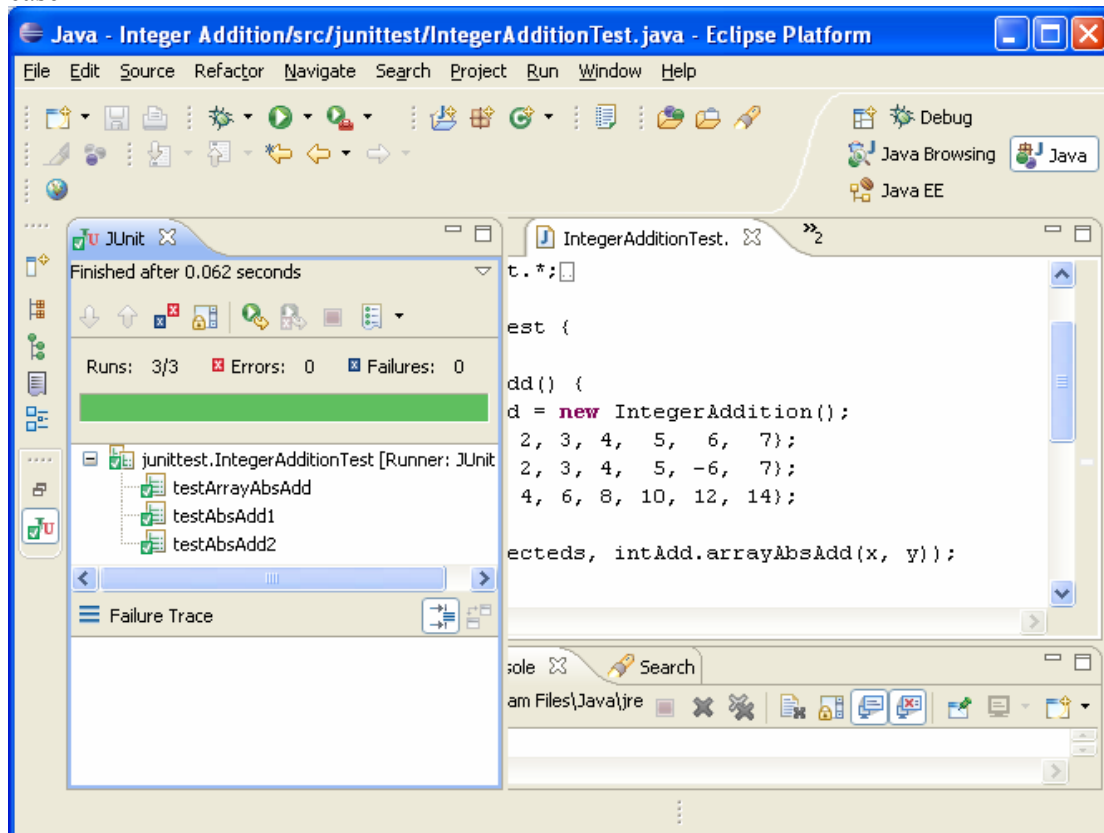
Trước khi kết thúc phần này, bạn hãy sửa lại code của hàm *absAdd* thành như sau:



```
package junittest;

public class IntegerAddition {
    public int absAdd(int x, int y) {
        return Math.abs(x) + Math.abs(y);
    }
}
```

Phương thức *abs()* là một phương thức static của lớp *Math* dùng để trả về giá trị tuyệt đối của một số. Bây giờ hãy chạy lại JUnit, bạn sẽ thấy kết quả trả về là đã pass được tất cả các test case



12. Lời kết

Như vậy bạn đã cùng tác giả đi hết bộ tutorial về Eclipse này. Tất cả những điều trình bày trong bộ tutorial này chỉ là một phần nhỏ của Eclipse. Tác giả hy vọng đã cung cấp cho bạn đủ những điểm cần thiết để có thể bắt đầu tự khám phá những điều thú vị khác.

Bộ tutorial này chắc chắn vẫn còn nhiều thiếu sót. Tác giả chân thành mong nhận được sự góp ý của tất cả mọi người, đặc biệt là những developer làm việc thực sự trong môi trường công nghiệp. Mọi ý kiến đóng góp xin gửi về địa chỉ hongtrungdung@gmail.com. Xin cảm ơn.