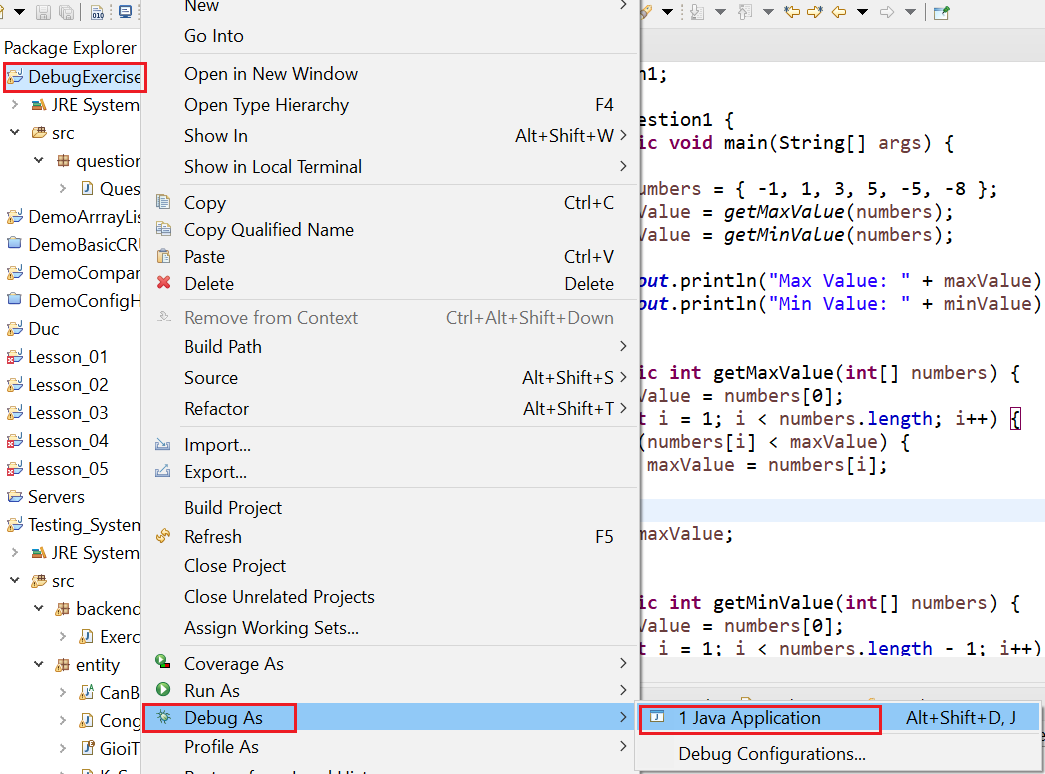
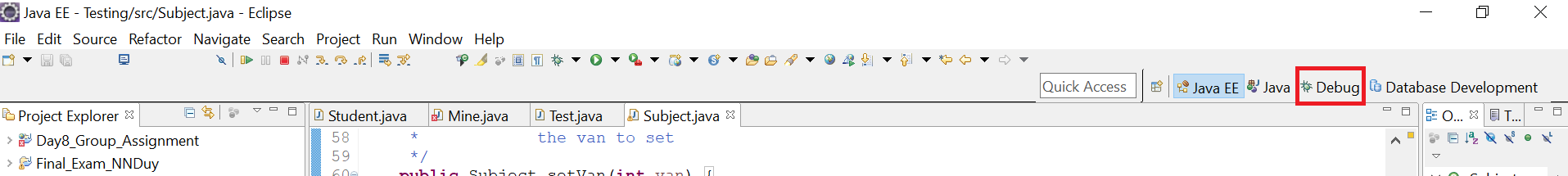
1. **Debug**
   1. **Cách vào mode debug**



*Phím tắt: Alt + Shift + D + J*

Nếu Eclipse không tự chuyển sang mode debug thì có thể click vào đây



* 1. **Các mode**
* **Step into (F5)**

Chạy tiếp vào trong method

* **Step return (F7)**

Thoát ra khỏi method vừa chọn ở F5

* **Step over (F6)**

Chạy xuống dòng tiếp theo (không vào trong method như F5)

* **Resume (F8)**

chạy tiếp tới breakpoint tiếp theo (nếu không có thì sẽ chạy hết chương trình)

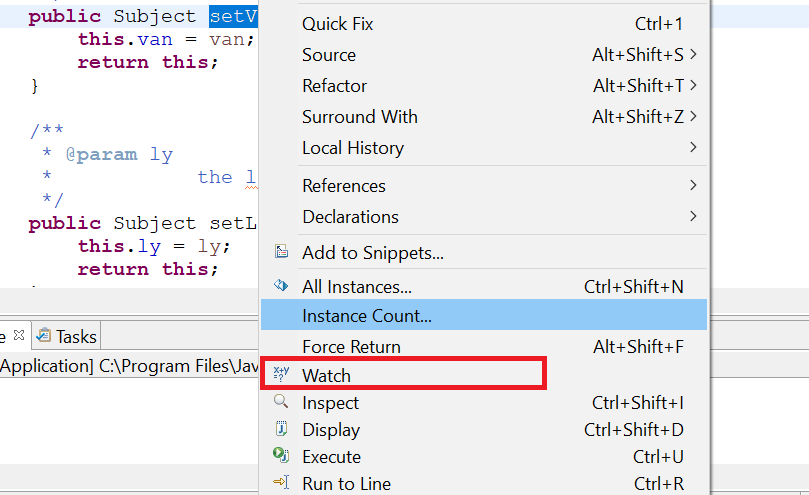
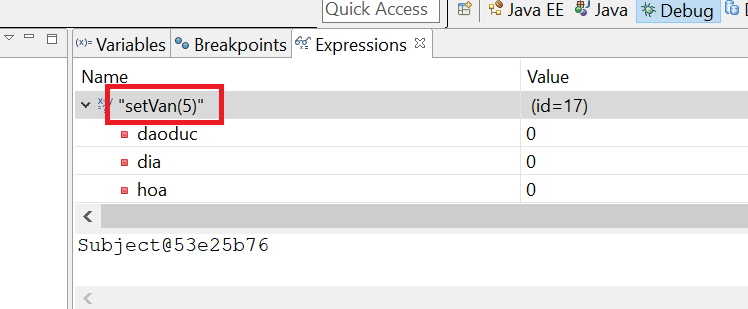
* **Stop (Ctrl + F2)**

Dừng debug lại

* 1. **Expression**

Có thể thực hiện biểu thức ngay tại lúc debug

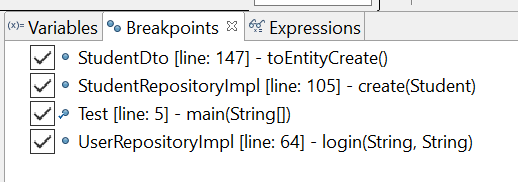
Bôi đen đoạn code muốn thực thi và click chuột phải vào **watch** để thực thi

*Có thể tự viết expression*

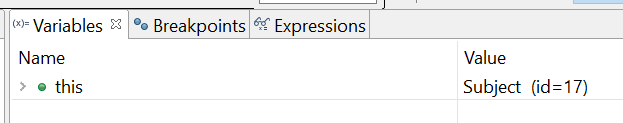
* 1. **Breakpoint**

Quản lý breakpoint tại tất cả project



* 1. **Variable**

Quản lý variable của context hiện đang debug



1. **Exception**

* Là 1 class java cung cấp để xử lý lỗi
* Phân loại
  + **Compile error**

Lỗi cú pháp, lỗi biên dịch

* + **Logical error**

Lỗi logic xảy ra khi chương trình đang chạy và làm chết chương trình

* **Cú pháp**

**try** {

// statement

}**catch** (SubException e) {

// statement when error

} **catch** (ParentException e) {

// statement when error

} **finally** {

// statement made last

}

*Nếu có nhiều catch thì cần để Super Exeption ở dưới Sub Exeption*

Note:

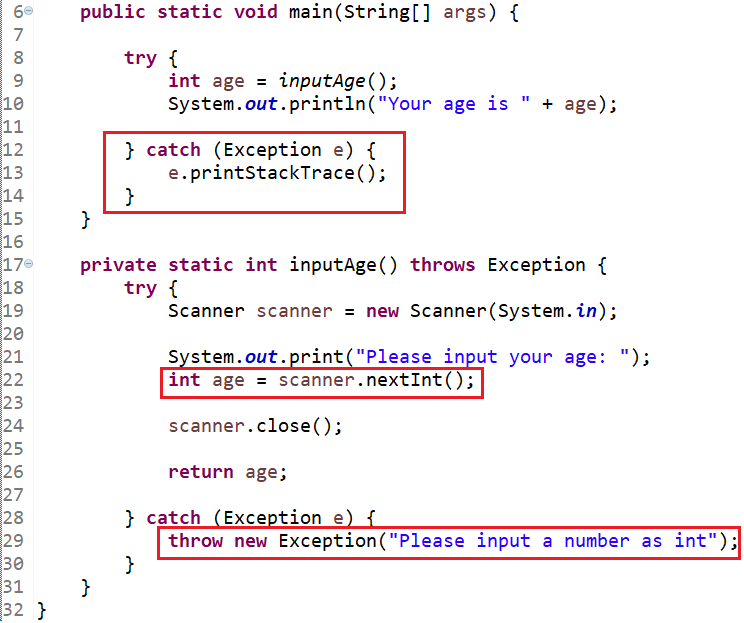
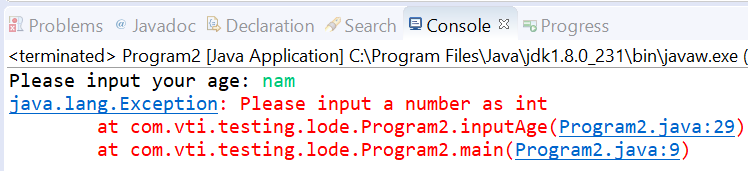
* + Những statement có khả năng bị lỗi thì phải cho vào try{}
  + Khi 1 statement bị lỗi thì nó sẽ tìm xem có catch nào xử lý lỗi đó ko (VD: InputMisMatchException), nếu không tìm thấy nó sẽ throw lên method cha
  + Những statement đặt trong finally được thực hiện ngay cả khi có lỗi hoặc không, thường dùng để đóng kết nối database

|  |  |
| --- | --- |
|  |  |

* **Throw và Throws**
  + Throw: "quăng ra", có thể "quăng" ở bất kỳ dòng nào trong phương thức.

Nếu method đó có khả năng xử lý lỗi thì ta xử lý luôn bằng cách đặt try catch, nếu method đó không có khả năng xử lý thì ta sẽ ném lên method cha để xử lý bằng cách throw lên trên

* + Throws: "Ném đi", chỉ có phương thức mới sử dụng throws, cảnh báo method cha là method con có lỗi, method cha bắt buộc phải xử lý lỗi của method con

** **

*Tìm hiểu thêm:* [*http://vietjack.com/java/tu\_khoa\_throws\_trong\_java.jsp*](http://vietjack.com/java/tu_khoa_throws_trong_java.jsp)

1. **Stack trace**

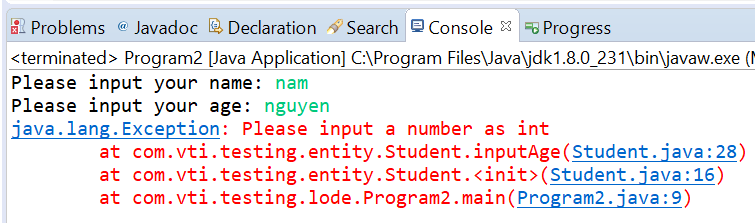
* Ngăn xếp Stack Trace mô tả cơ chế thực thi các method và xử lý exception
* Các method được thực thi theo cơ chế stack (LIFO)

Nếu 1 line gặp lỗi thì sẽ tìm kiếm bất kì xử lý lỗi nào trong method (VD: try…catch).

Nếu không có bất kì xử lý lỗi nào (không có try catch) thì nó sẽ throw lỗi lên parent method

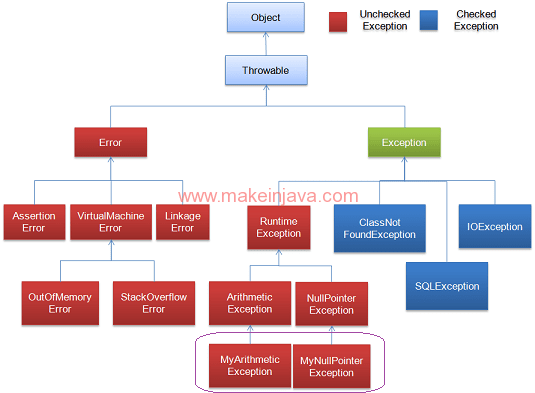
Nếu parent cũng không có xử lý lỗi nào thì exception sẽ được đẩy lên thông qua stack 🡺 thông báo lỗi được in ra cửa sổ Output

Nếu không gặp lỗi thì method sẽ bị remove đi khỏi stack trace

****

1. **Phân biệt Error và Exeption**
   1. **Error**

* Là chỉ ra các lỗi nghiêm trọng và điều kiện bất thường mà các ứng dụng không mong đợi.
* Developer không thể bắt và xử lý khi gặp error.
* Các error phổ biến là outOfMemory, lỗi phần cứng,…
  1. **Exeption**
* Là các ngoại lệ xảy ra trong code.
* Developer có thể bắt và xử lý lỗi nếu cần thiết thông qua **try catch** hoặc **throws** lên tầng trên và xử lý trên tầng trên.



* Có 2 loại exception là
  + **Check exceptions**

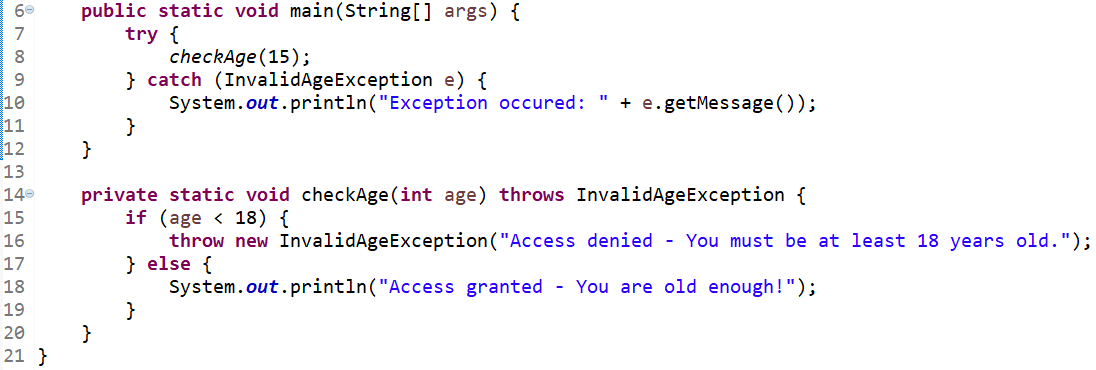
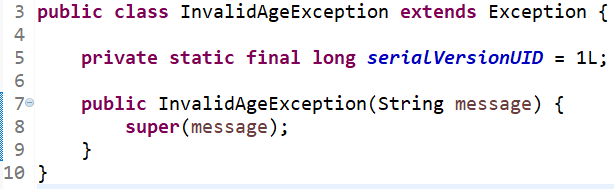
Là các ngoại lệ được kiểm tra tại compile time. Khi gặp exepion này phải try catch hoặc throws exception.

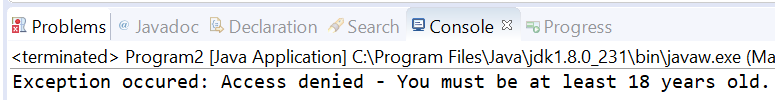
* + **Uncheck exception**

Là ngoại lệ kiểm tra tại run-time.

1. **Custom Exception**

Ta có thể tự tạo ra exception của mình, sau này những logic như nhập tuổi, nhập email, nhập sđt, ... logic exception ta nên tự tạo ra custom exception riêng



* **Reference**