

4. Class of Classes

Object-Oriented Design & Programming
2016년도 봄학기

도입부

- 뭔가 이상한 실습 분위기 아래에서도 지금까지 근성 있게 따라와 준 친구들, 고맙습니다.

도입부

- 이제 오늘부터,
본격적인 OODP 개념들을
실습에서 다루기 시작할 예정입니다.

도입부

- 이제 오늘부터,
본격적인 OODP 개념들을
실습에서 다루기 시작할 예정입니다.

도입부

- 쉽게 말하면,
오늘부터 다양한 클래스들을 만듭니다.
- 이제부터 실습 한 주 빠지면 손해가 크니
능동적인 일정 관리를 통해
최대한 실습에 원만히 참여해 봅시다.
 - 미리 연락해서 다른 반 수업 참여 가능
 - 물론 개인별 조교 면담도 가능

이번 시간에는

- Eclipse의 debugger 사용 방법
- Class of classes
 - Data를 위한 class와 code를 위한 class
 - 가장 간단한 data element class 만들어 보기
- 실습과제#3

Debugger 사용 방법

- 시작하기 전에,
디버그를 시도해 볼 Java project를
미리 하나 골라 봅시다.
 - 조교는 HY-in 학생자료실에 있는
OODP_HW_2_3_by_TA.zip을 import해 쓸 예정입니다.
 - '자신이 작성한 project'를 고르는 것이 가장 좋지만
여의치 않은 경우 조교와 같은 project를 사용해 봅시다.

Debugger 사용 방법

- 주요 디버그 시나리오
 - 무한루프가 어디서 도는지 알고 싶을 때
 - 빨간 오류가 났지만 원인을 찾기 어려울 때
 - 반복문의 특정 번째 실행을 확인하고 싶을 때

Debugger 사용 방법

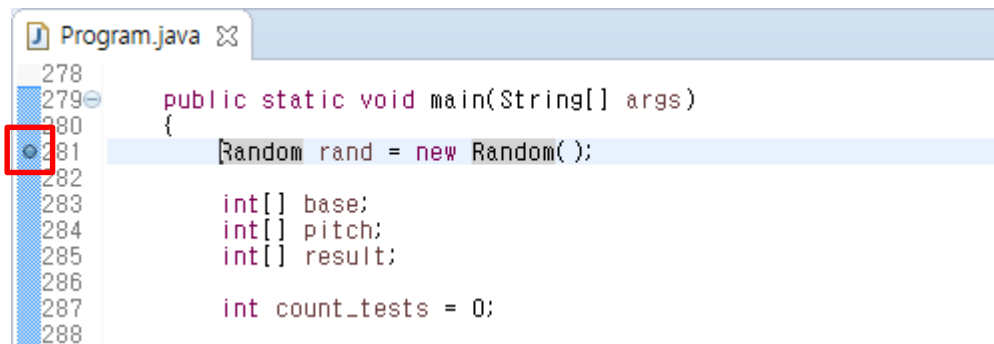
- 기본 디버그 작업
 - 실행 중단하기
 - 한 단계씩 실행하기
 - 현재 변수에 든 값 확인하기

Debugger 사용 방법

- 준비가 되었으면, 다음 슬라이드부터 Eclipse debugger의 기본 사용 방법을 차근차근 살펴 봅시다.

Debugger 사용 방법

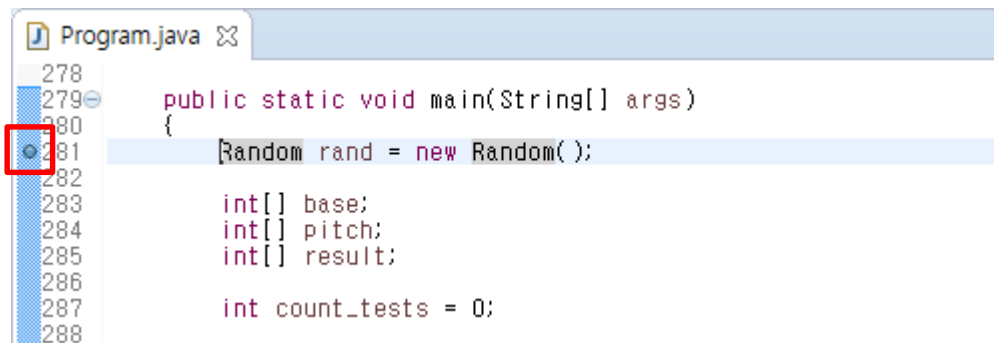
- Ctrl + Shift + B: Breakpoint 놓기
 - 코드 창 현재 줄 왼 쪽에 푸른 점이 보이면 성공!
 - 우선 지금은 main() 중괄호 안 첫 줄에 놓읍시다.



Debugger 사용 방법

- Breakpoint

- 디버그 모드로 실행할 때,
이 파란 점을 만나면 실행을 중단합니다.
 - 그래서, 당연히, '실행되지 않는 줄'에는 놓을 수 없습니다.
(예: 메서드 밖)

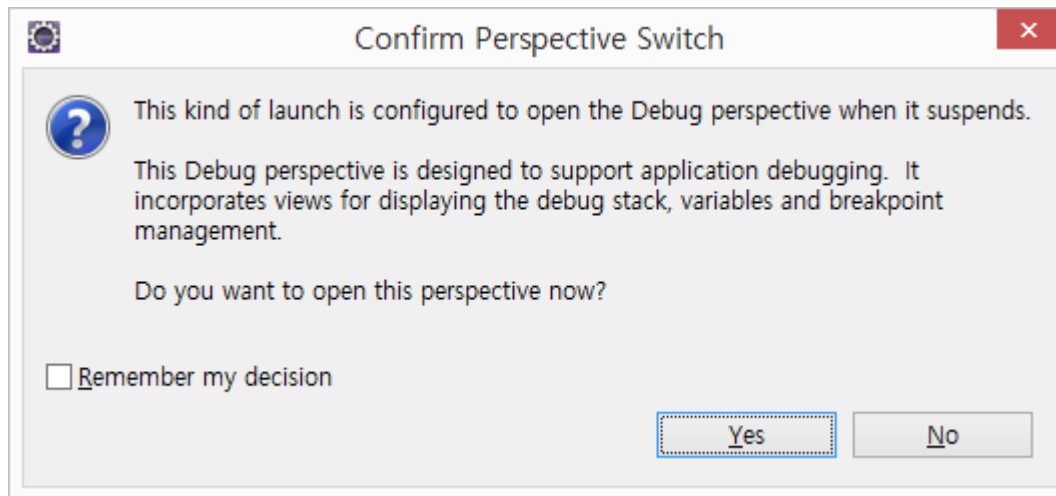


Debugger 사용 방법

- F11: 디버그 시작

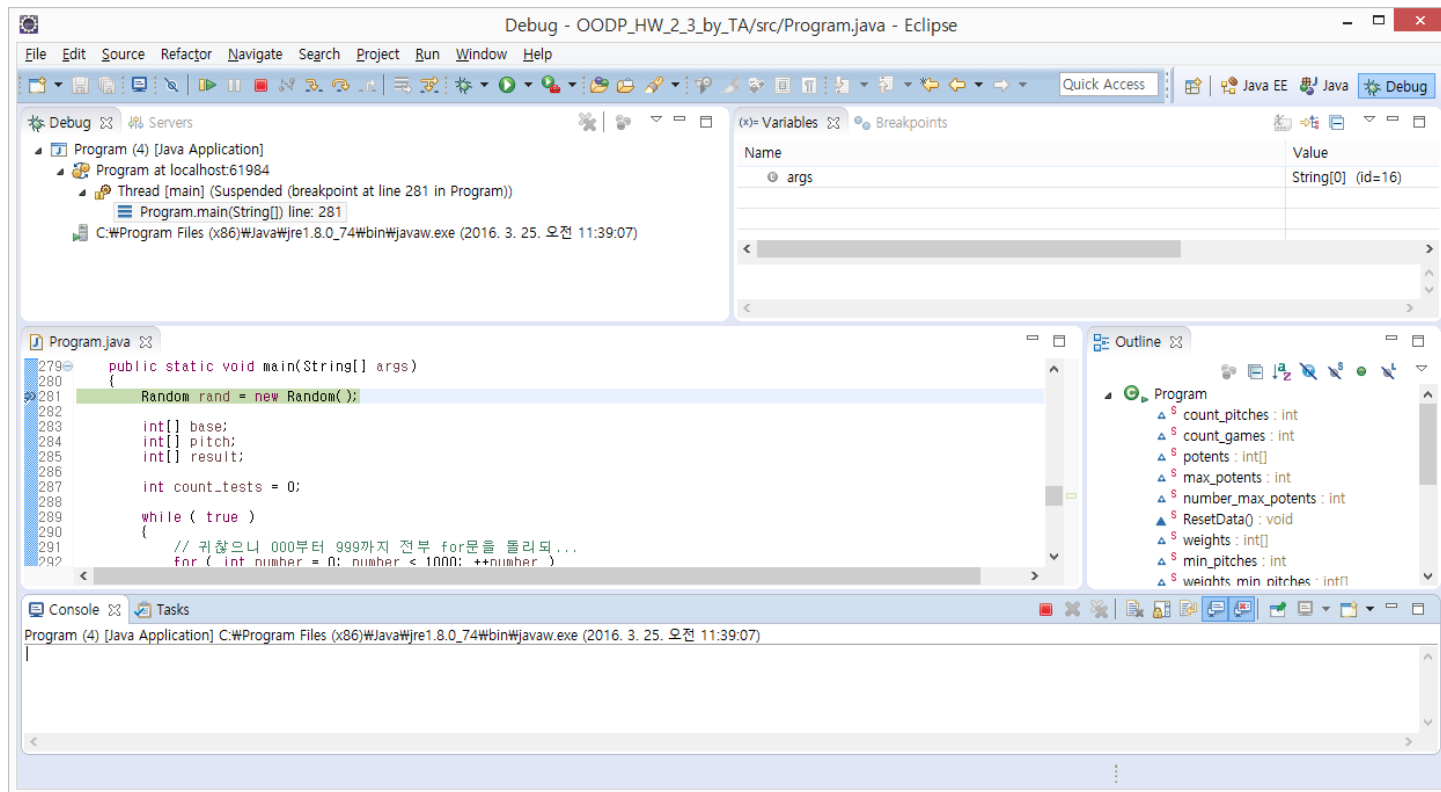
Debugger 사용 방법

- F11: 디버그 시작
 - 처음 breakpoint를 놓아 본 친구들은 이 창을 처음으로 만나 볼 수 있습니다.
 - '디버그용' UI로 가겠냐는 것이니 Yes를 눌러 줍시다.



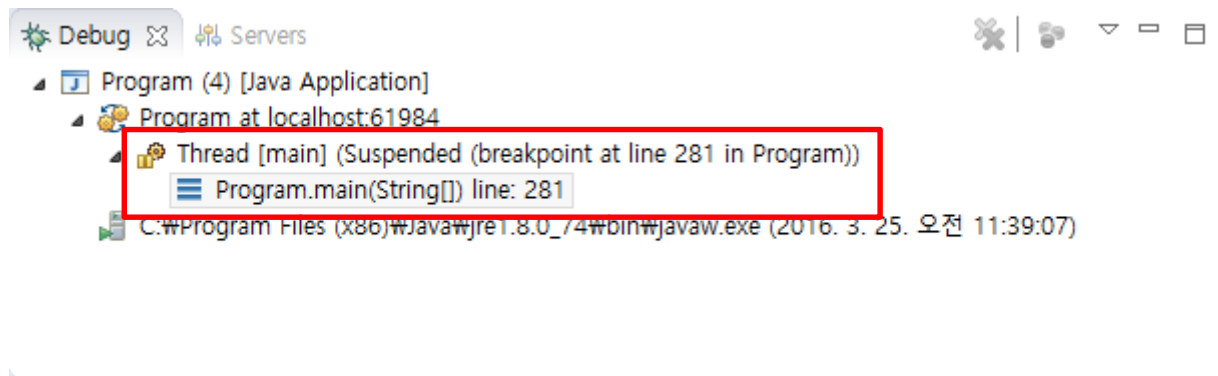
Debugger 사용 방법

- 복잡한 탭들이 뜨면 성공!



Debugger 사용 방법

- Debug 탭
 - 현재 실행중인 Java 프로그램 목록이 뜹니다.
 - 이 중 하나는 우리가 방금 중단한 그 친구입니다.
 - 모르는 단어들은 일단 그냥 그러려니 하되,
아래 보이는 내용이 'call stack'인 것은 알아 둡시다.



Debugger 사용 방법

- Variables 탭

- 현재 보이는 지역 변수들의 목록을 보여 줍니다.
- 실행이 진행되면서 종종 바뀝니다.
- 사실 이름에 마우스 갖다 대도 값이 보이기 때문에 그리 자주 쓰이는 탭은 아닙니다.



Debugger 사용 방법

- 단계별 실행
 - F5: Step into(꼼꼼하게 실행)
 - F6: Step over(대강대강 실행)
 - F7: Step return(이 메서드가 끝날 때까지 쭉 실행)
 - F8: Resume(다음 breakpoint까지 쭉 실행)

Debugger 사용 방법

- 단계별 실행

- F5: Step into(꼼꼼하게 실행)

- F6: Step over

실행이 메서드를 잠시 떠날 때(다른 메서드 실행 등),
debugger도 함께 따라갑니다.

- F7: Step return

- F8: Resume(다음 breakpoint까지 쭉 실행)

Debugger 사용 방법

- 단계별 실행

- F5: Step into(꼼꼼하게 실행)

- F6: Step over(대강대강 실행)

- F7: Step return

- F8: Resume()

실행이 메서드를 잠시 떠나도
debugger는 현재 메서드에 남아 있으며,
실행은 보통 '이 메서드의 다음 줄'에서 다시 멈춥니다.

Debugger 사용 방법

- 단계별 실행

- F5: Step into

- F6: Step over

- F7: Step return(이 메서드가 끝날 때까지 쭉 실행)

- F8: Resume(다음 breakpoint까지 쭉 실행)

이 아래 친구들은 여기 적힌 설명 그대로 동작합니다.

Debugger 사용 방법

- F5같은 평션 키가 작은 친구들은 디버그 UI 상단에 있는 메뉴 버튼들을 클릭해도 됩니다.
 - 마우스 갖다 대면 이름 나오니 설명은 생략

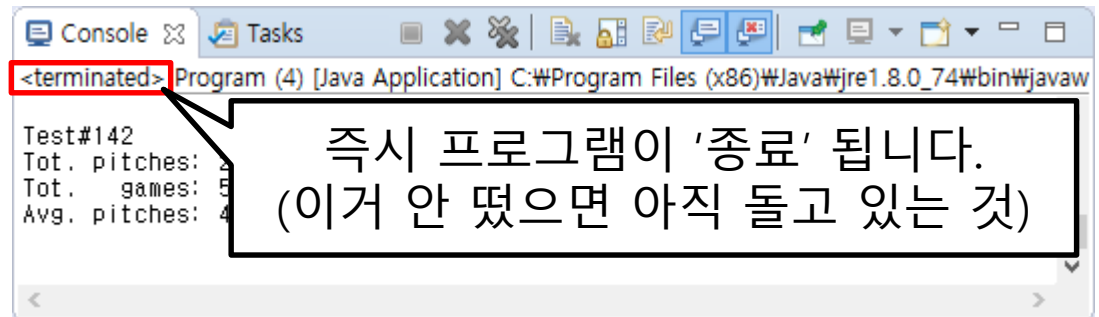
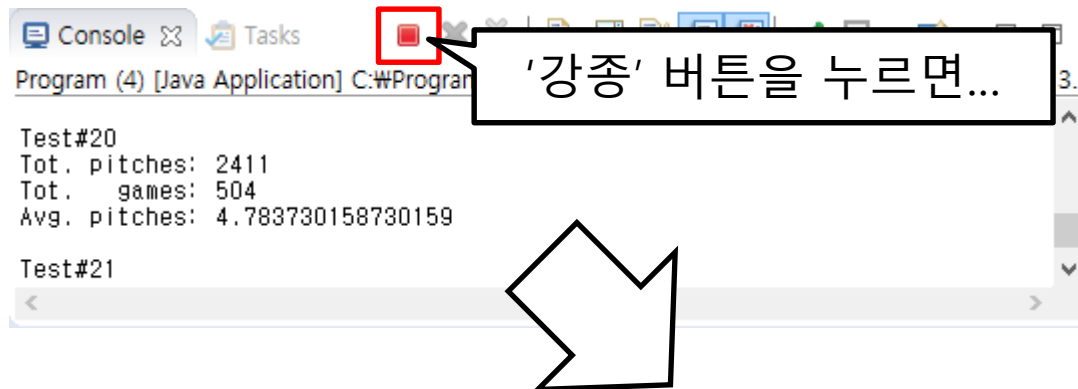


Debugger 사용 방법

- 디버그를 종료할 때는...
 - 아직 프로그램이 돌고 있는 경우 '강종'시키고
 - Java perspective(UI 모드)로 돌아오면 됩니다.

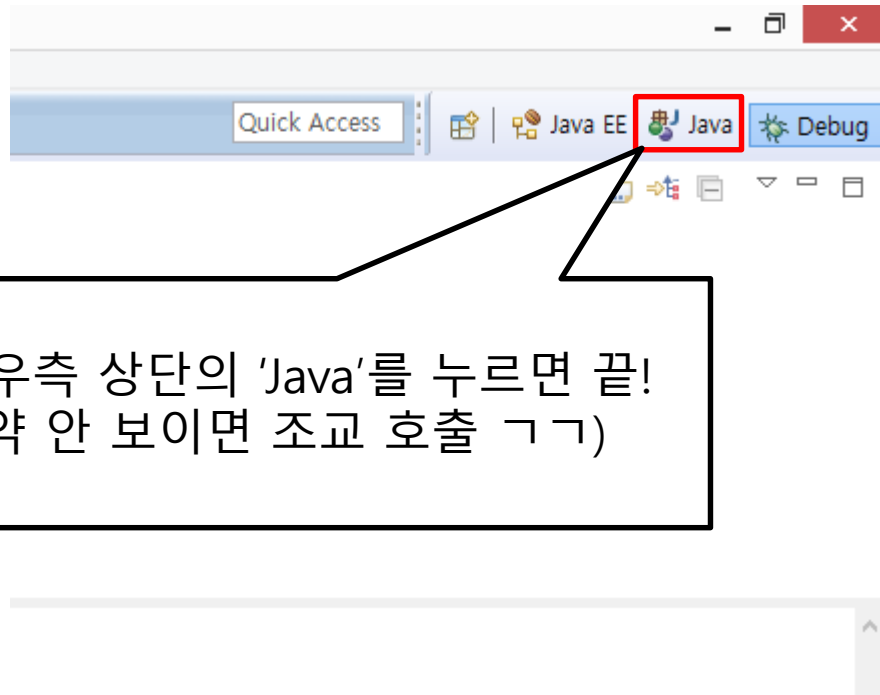
Debugger 사용 방법

- '강종' 방법



Debugger 사용 방법

- Java perspective로 돌아오기



화면 우측 상단의 'Java'를 누르면 끝!
(만약 안 보이면 조교 호출 ㄱㄱ)

Debugger 사용 방법

- 이 정도 알아 두면
기본적인 debugger 사용 방법은 끝!

Debugger 사용 방법

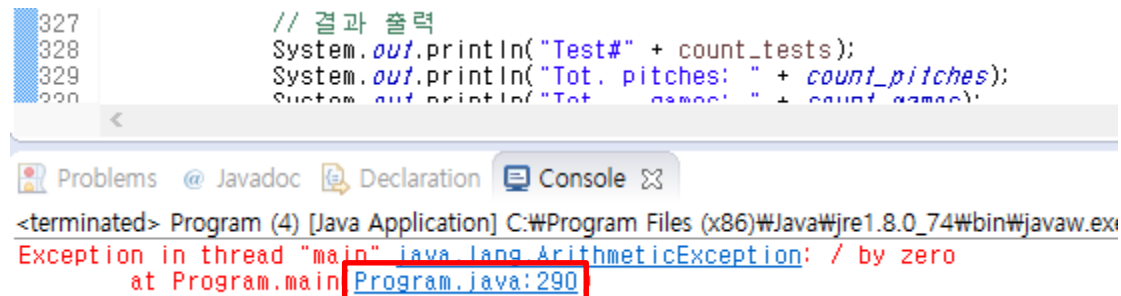
- 주요 디버그 시나리오
 - 무한루프가 어디서 도는지 알고 싶을 때
 - 빨간 오류가 났지만 원인을 찾기 어려울 때
 - 반복문의 특정 번째 실행을 확인하고 싶을 때

Debugger 사용 방법

- 시나리오별 디버그 방법
 - 무한루프일 때
 - 일단 main() 맨 윗 줄에 breakpoint를 놓고
 - F11 눌러 디버그를 시작한 다음
 - 일단 F8을 눌러 실행을 좀 진행시키다가
 - 메뉴의 resume 버튼 옆에 있는 '일시정지'를 누르면, 방금까지 실행되던 위치에서 프로그램이 정지합니다.
 - 이제 일단 '여기가 무한루프 안'인 것은 알았으니 **적절한 운영과 적절한 마무리로 GG를 받아 내시면 됩니다.**

Debugger 사용 방법

- 시나리오별 디버그 방법
 - 빨간 오류의 원인을 찾기 어려울 때
 - 일단 오류 메시지의 코드 링크를 클릭해 봅니다.
 - 링크가 여러 개면 맨 위에 있는 것을 클릭



The screenshot shows a code editor with the following Java code:

```
327 // 결과 출력
328 System.out.println("Test#" + count_tests);
329 System.out.println("Tot. pitches: " + count_pitches);
330 System.out.println("Tot. asmc: " + count_asmc);
```

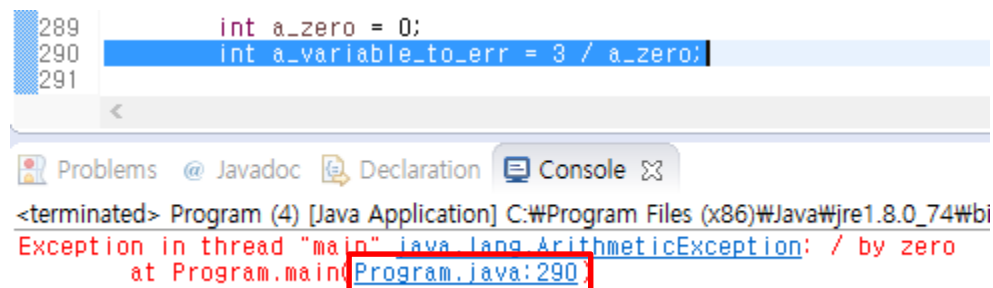
Below the code editor, the console window displays the following error message:

```
<terminated> Program (4) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_74\bin\javaw.exe
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Program.main Program.java:290
```

The text "Program.java:290" is highlighted with a red box, indicating the location of the error.

Debugger 사용 방법

- 시나리오별 디버그 방법
 - 빨간 오류의 원인을 찾기 어려울 때
 - 일단 오류 메시지의 코드 링크를 클릭해 봅니다.
 - 링크가 여러 개면 맨 위에 있는 것을 클릭
 - 오류가 발생한 그 줄로 이동할 수 있습니다.



The screenshot shows an IDE with a code editor and a console window. The code editor displays the following lines:

```
289 int a_zero = 0;  
290 int a_variable_to_err = 3 / a_zero;  
291
```

The console window shows the following error message:

```
<terminated> Program (4) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_74\bin\java.exe  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at Program.main(Program.java:290)
```

The error message is highlighted in red, and the line number 290 is highlighted in blue. The code editor also has line 290 highlighted in blue.

Debugger 사용 방법

- 시나리오별 디버그 방법
 - 빨간 오류의 원인을 찾기 어려울 때
 - 일단 오류 메시지의 코드 링크를 클릭해 봅니다.
 - 링크가 여러 개면 맨 위에 있는 것을 클릭
 - 오류가 발생한 그 줄로 이동할 수 있습니다.
 - 이제 일단 '여기가 오류를 발생시킨 곳'인 것은 알았으니 **적절한 운영과 적절한 마무리**로 GG를 받아 내시면 됩니다.

Debugger 사용 방법

- 시나리오별 디버그 방법
 - 반복문의 특정 루프를 확인하고 싶을 때
 - '이번 루프가 내가 원하는 루프인가'를 검사하는 조그만 if문을 반복문 시작 부분에 적어 줍니다.
 - 방금 만든 if문 안에 간단한 sysout을 적어 줍니다.
 - 방금 적은 sysout에 breakpoint를 놓습니다.
 - F11을 눌러 디버그를 시작합니다!

Debugger 사용 방법

- Q: 설명이 좀 애매한데요...

Debugger 사용 방법

- Q: 설명이 좀 애매한데요...
 - 오류의 원인이 매 번 다르기 때문에,
이를 고치기 위한 디버그 방법 또한 몹시 다양합니다.
 - 그래도, 작은 프로그램 만들 때 디버그 방법을 숙달해 두면
나중에 '이해조차 안 되는 프로그램' 만들 때 도움이 됩니다.
 - 그나마 우리 실습에서는 조교가 옆에 있으니,
여러분이 막혔을 때 혼자 디버그하다 안 되면
꼭 조교의 도움을 요청해 주세요.
 - 물론 가장 좋은 방법은 '오류를 안 내는 것'입니다.
코드 작성은 집중력이 남아 있을 때만 ㄱㄱ해 주세요.

Debugger 사용 방법

- Q: 사용 방법이 이게 다인가요...?

Debugger 사용 방법

- Q: 사용 방법이 이게 다인가요...?
 - 더 치명적인 기능들이 몇 개 더 있습니다.
- 하지만 지금 우리 수준에선
그 기능들은 알맞게 쓰기 어려우니(그리고 귀찮음)
관심 있는 친구들은 조교에게 천천히 물어봐 주세요.

실습과제#3

- 로봇 장난감을 클래스로 만들어 보기
 - 총 세 개의 부분과제로 구성되어 있습니다.
 - Debugger 사용법 익히기 (60점)
 - Data element class 만들기 (30점)
 - Data element class 좀 더 꼼꼼하게 만들기 (10점)
- 각자의 선택에 따라 과제를 수행한 다음,
과제 페이지에 느낀 점을 적어 제출하세요.
 - 제출기한: 오늘 23:59:59까지

실습과제#3

- 로봇 장난감을 클래스로 만들어 보기
 - 이제 잠시 쉬는 시간을 가지면서
실습과제#3-1을 각자 수행해 보도록 합시다.

Class

- 지난 시간에 말했던,
Program = Data + Code를 기억하나요?

Class

- (과거의)C까지만 해도,
data와 code가 언어적으로 분리되어 있어,
몇몇 요소를 만들 때 꽤 귀찮았습니다.
 - Data와 이를 유지하기 위한 code가 따로 정의됨
 - 예) FILE*와 fopen()
 - Code와 이를 돌리기 위한 data가 따로 정의됨
 - 예) fprintf()와 FILE*

Class

- 귀찮다는 것은...
 - '못 만든다'는 것은 아닙니다.
 - Java로 작성 가능한 프로그램은 C로도 얼마든지 작성할 수 있습니다.
 - 다만, 제대로 그 요소들을 사용하려면, data와 code의 상관 관계를 프로그래머가 반드시 기억해 두고 있어야 합니다.
 - fopen()이야 자주 쓰니 익숙하겠지만, 여튼 기억할 게 많다는 것은 정말 귀찮은 일입니다.

Class

- Class는 기본적으로...
 - Data와 이를 유지하기 위한 code를 한 곳에
 - Code와 이를 돌리기 위한 data를 한 곳에
 - ...두기 위해 만들어진 개념입니다.

Class

- 잘 만든 class를 쓰는 것은 덜 귀찮습니다.
 - 우리가 쓰면 안 되는 기능을 못 쓰게 막기 때문입니다.
 - 함부로 막 만들면 안 되는 data를 못 만들게 함
 - Data 발급용 내부 code를 못 보게 함
 - Code 실행에 필수적인 data를 못 보게 함
- 우리한테는 '써도 되는 것'만 보이기 때문에,
(과거의)C에 비해 상대적으로 기억할 것이 적고,
그래서 덜 귀찮습니다.

Class

- 물론 반대로,
class를 잘 만드는 것은 **정말** 귀찮습니다.
 - 앞으로 이 class를 누가 어떻게 쓸 것인지 예상하고 적절한 키워드를 써 가며 막아 두어야 합니다.
- 심지어 우리가 실습 때 작성하는 class들의 99.99%는 '나 빼고 아무도 안 쓸 것'이 거의 확실하며, 이러한 사실은, 여러분을 욕구와 이상향 속에서 갈등하게 만듭니다.

Class

- 그래도, 조금은 자비로운 마음을 가지고, 다양한 class들을 '잘' 만들어 보도록 열심히 노력해 보세요.
- 나중에 큰 프로그램 만들 때는
'내가 내 class를 쓰는 빈도' 또한 몹시 높아집니다.
그런 경우, 미리 작은 class들을 잘 만들어 두면
'잠시 뒤의 여러분'에게 큰 도움이 될 수 있습니다.

Class

- 그래도, 조금은 자비로운 마음을 가지고, 다양한 class들을 '잘' 만들어 보도록 열심히 노력해 보세요.
- 물론 우리가 지금 당장 큰 과제를 할 순 없겠지요? 그럼에도 불구하고, 마음만은 '국가 프로젝트'를 하듯 결의를 다지고 일종의 role-play를 시도해 보기를 권장합니다.
 - 여러분 과제에 대한 CTO가 되어 보세요!

Class of classes

- 우리 주변에는 다양한 class들이 있습니다.

Class of classes

- 우리 주변에는 다양한 class들이 있습니다.
 - System class: 실행에 필요한 핵심 요소들의 모임
 - Scanner class: 입력을 잘 다루기 위한 '메서드' 집합
 - Random class: 랜덤 '값'의 원천

Class of classes

- 우리 주변에는 다양한 class들이 있습니다.
 - 이들은, 각자 쓰임새가 다른 만큼 서로 다른 스타일로 만들어져 있습니다.
- 우리 실습에서는 class의 종류를 크게 네 가지로 나누고 이들 각각에 대한 구현 방법을 몇 주 동안 천천히 살펴 볼 계획입니다.

Class of classes

- Data를 중시하는 class
- Code를 중시하는 class (이건 다음에)

Class of classes

- Data를 중시하는 class
 - Data element class
 - 'Collection' class (이것 역시 다음에)

Class of classes

- Data를 중시하는 class
 - Data element class
 - 말 그대로, 의미 있는 데이터 묶음입니다.
 - 여러분이 작년에 배운 '구조체'와 가장 유사합니다.
 - 차이점은... 아까 설명했으니 대강 느낌이 오겠지요?

Class of classes

- Data element class 만들어 보기
 - 이제 조교와 함께 실습과제#3-2를 진행하면서...
 - Java에서 새 class를 만드는 방법(사실 Program class와 동일)
 - 필드 선언 방법
 - '생성자' – 새 인스턴스를 자동으로 초기화하는 방법
 - 내 인스턴스를 sysout으로 쉽게 출력할 수 있게 만드는 방법
 - ...을 살펴 봅시다.

오늘 내용 정리

- Debugger를 통해 실행을 살펴볼 수 있다.
- 클래스의 쓰임새는 다양하다.
 - 오늘 한 가지를 배웠고 앞으로 더 배울 예정
- Data element class는 구조체와 유사하다.
 - '생성자', toString(), 각종 조작용 메서드들을 더해 씀