2019 소프트웨어프로젝트

Project 1

프로그램 수행시간 측정

중앙대학교

창의ICT공과대학 소프트웨어학부

20185659 김혜성

목차

Table of contents

1. Abstract

이 프로젝트에서는 자바 실행 모델과 Eclipse 사용법을 익히는데 목적을 두어 '프로그램 수행시간 측정'을 진행하였다.

프로젝트 수행 결과는 다음과 같다.

1. a^2을 계산을 계산하는 과정에서 지수승을 사용하는 계산 방법이 $a \times a$ 로 계산하는 방법보다 시간적인 측면에서 덜 효율적이다.
2. 반복적인 수행 결과, 측정값의 편차가 있다.

이 paper에서 프로젝트에 대한 고찰은 다음과 같다.

1. 이 프로젝트에서 수행시간이 '0'인 경우가 나타나지 않은 원인이 무엇인가?
2. 수행시간이 짧아서, nano-second 보다 작다면 어떻게 측정할 것인가?
3. 측정값의 편차가 크다면 어떻게 보정할 것인가?

2. Introduction

이 프로젝트의 목적은 다음과 같다.

1. Java 프로그램 입문 : 프로그램 작성, 컴파일, 실행실습
   - 자바 실행 모델
2. Eclipse 사용법

위 두가지 사항을 목적으로 두어 '프로그램 수행 시간 측정' 을 주제로 프로젝트를 진행하였다.

프로젝트의 내용은 다음과 같다.

a^2을 계산하는 방법은 지수승을 사용하는 방법과 $a \times a$ 로 계산하는 방법이 있다.

이 프로젝트에서는 두 가지 방법 중 어떤 것이 시간적인 측면에서 효율적인지를 검증하는 Java 프로그램을 작성하였다.

이 프로그램을 작성하는데에 사용된 method는 다음과 같다.

1. 시간 측정 방법
   - 계산 전 시간 측정
   - 계산 후 시간 측정
   - 계산 전과 후의 시간의 차이 계산
2. 관련 Java method
   - 지수승 계산 : [1] Meth.pow(a, 2)
   - 현재 시간 : [2] long System.nanoTime( )
   - 결과 출력 : [3] System.out.printIn( )

a^2을 두 가지 방법으로 계산하고 이에 소요된 시간을 측정하여 츨력하였다.

---

[1] public static double pow(double a, double b) : Returns the value of the first argument raised to the power of the second argument. [1]

[2] public static long nanoTime( ) : Returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds. [2]

[3] public static final PrintStream out : The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user. [2]

3. Proposed Program

3.1. Algorithm

프로젝트1은 a라는 변수에 대하여 a^2을 계산하는 데에 소요된 시간을 측정하는 Java 프로그램이다.
프로그램의 알고리즘은 다음과 같다.

a ← 3	▷ declaration of the number to calculate
System.nanoTime( ) result ← multiplication of 'a' System.nanoTime( ) display the output : endTime - startTime	▷ returns the current value of the time for the startTime ▷ $a \times a$ ▷ returns the current value of the time for the endTime ▷ calculate the execution time
System.nanoTime( ) result ← exponentiation of 'a' System.nanoTime( ) display the output : endTime - startTime	▷ returns the current value of the time for the startTime ▷ a^2 ▷ returns the current value of the time for the endTime ▷ calculate the execution time

3.2. Compile and Build with cmd

자바로 프로그램을 개발하려면 JDK이외에 메모장(notepad.exe)이나 에딧플러스(editplus)와 같은 편집기가 필요하다. [3]

```
public class ExecTime {

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

		int a = 3;
	long startTime = System.nanoTime();
	double result = a * a;
	long endTime = System.nanoTime();

	System.out.println("Execution Time of multipication = " + (endTime - startTime));

	startTime = System.nanoTime();
	double result2 = Math.pow(a,2);
	endTime = System.nanoTime();

	System.out.println("Execution Time of exponentiation = " + (endTime - startTime));

	}


}
```

▲ **Fig. 1.** Source code of the ExecTime.java with cmd

프로젝트1에서의 편집기는 **Fig. 1.** 에서와 같이 Vim을 사용하였다. Vim(Vi IMproved)은 vi 호환 텍스트 편집기이다. Vim 스크립트 등을 사용해서 자유롭게 편집 환경을 변경할 수 있다. [4]
이 프로그램을 실행하려면, 먼저 자바컴파일러(java.exe)를 사용해서 소스파일(ExecTime.java)로 부터 클래스파일(ExecTime.class)을 생성해야 한다. 그 다음에 자바 인터프리터(java.exe)로 실행한다. [3]

```
Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ vim ExecTime.java
Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ javac ExecTime.java
Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
Execution Time of multipication = 309
Execution Time of exponentiation = 28620
```

▲ **Fig. 2.** Procedure of the ExecTime.java with cmd

위의 **Fig. 2.**는 Mac OS에서의 자바 실행 모델이다.
프로젝트1을 수행하기 위한 사전 작업은 아래와 같다.
우선, JDK(Java Development Kit)를 설치했다. JDK를 설치하면, JVM(Java Virtual Machine)과 자바클래스 라이브러리 등의 프로그램이 설치된다.

JDK를 설치한 후, 설치된 디렉토리의 bin 디렉토리를 path에 추가해주어야 한다. Path는 OS가 파일의 위치를 파악하는데 사용하는 경로로, path에 추가함으로써 해당 디렉토리에 포함된 파일을 파일 경로없이 파일 이름만으로도 사용할 수 있게 된다. [3]
자바 실행 모델에 사용된 JDK의 주요 실행파일들은 다음과 같다.
- javac.exe : java comliler. 자바 소스코드를 바이트 코드로 컴파일한다.

```
Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ javac ExecTime.java
```

▲ **Fig. 3.** java compiler of the ExecTime.java with cmd

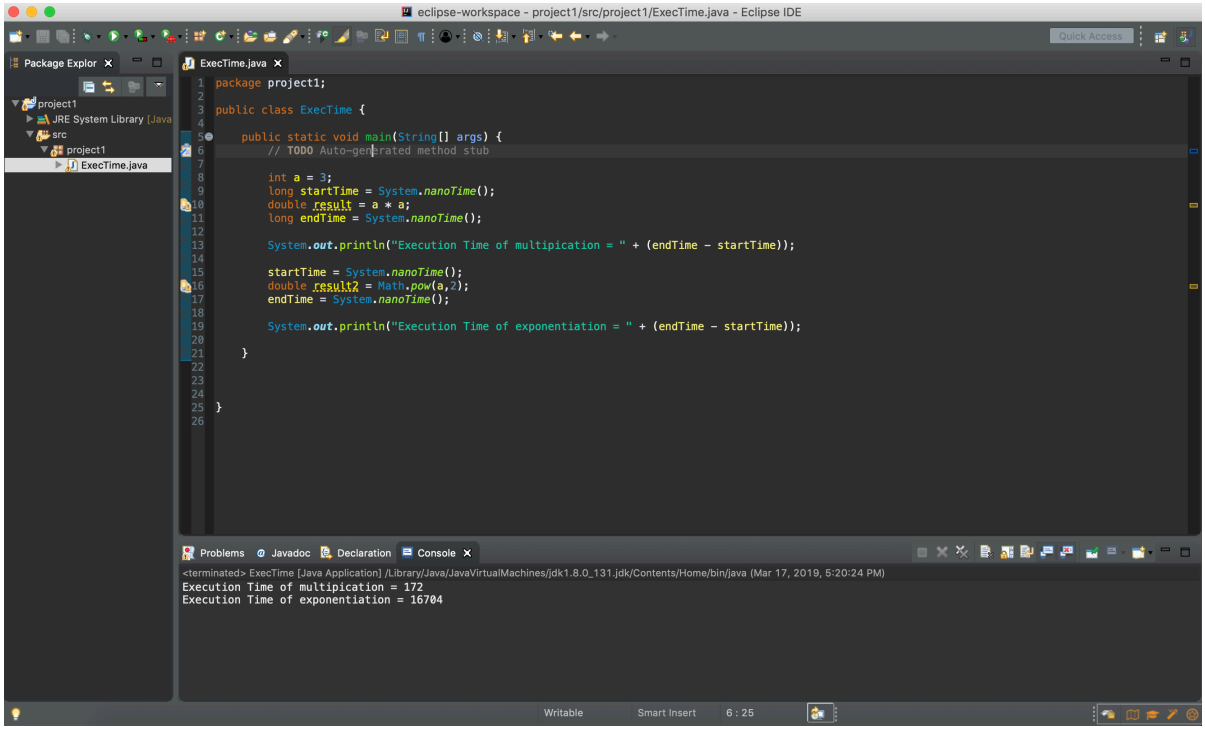
- java.exe : java interpreter. compiler가 생성한 바이트 코드를 해석하고 실행한다.

```
Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
```

▲ **Fig. 4.** java interpreter of the ExecTime.java with cmd

콘솔에서 **Fig. 4.**와 같이 Java 애플리케이션을 실행시켰을 때 내부적인 진행순서는 다음과 같다.
1) 프로그램의 실행에 필요한 클래스(ExecTime.class)를 로드한다.
2) 클래스파일을 검사한다. (파일형식, 악성코드 체크)
3) 지정된 클래스(ExecTime)에서 main(String[ ] args)를 호출한다.
main메서드의 첫 줄부터 코드가 실행되기 시작하여 마지막 코드까지 모두 실행되면 프로그램이 종료되고, 프로그램에서 사용했던 자원들은 모두 반환한다.[3]

3.3. Source Code with eclipse



▲ **Fig. 5.** Source Code of the ExecTime.java with eclipse

프로젝트1의 구성은 다음과 같다.
- project name : project1
- class name : ExecTime (static main( )을 포함한다.)
ExecTime.java의 source code를 간략하게 설명하자면 다음과 같다.
9: long startTime = System.nanoTime( ); // System class에 있는 method로, 현재 JVM의 hight-resolution 시간값을 nano sec. 단위로 반환한다. 단, 시스템이나 시각과는 아무런 연관성이 없다. [2]

이 method를 이용하여 시간을 측정했다. 이때, 시간 측정 자료형이 long으로 되어있으므로 long type으로 변수를 선언하였다.

10: long result = a * a; // 아래의 지수승 연산과정에서 method의 반환 type이 0이므로 이의 연산과정과 비교할 때 공평성을 두기 위해 long type으로 선언하였다.

16: double result2 = Math.pow(a, 2); // Math class에 있는 method로, 지수승 계산값을 반환한다. 이때, method의 반환 type은 double이므로 double type으로 변수를 선언하였다.

4. Experimental Results

```
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 309
 Execution Time of exponentiation = 28620
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 217
 Execution Time of exponentiation = 47569
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 332
 Execution Time of exponentiation = 25106
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 339
 Execution Time of exponentiation = 31875
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 225
 Execution Time of exponentiation = 25582
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 265
 Execution Time of exponentiation = 25435
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 255
 Execution Time of exponentiation = 17355
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 162
 Execution Time of exponentiation = 21670
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 414
 Execution Time of exponentiation = 24551
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 337
 Execution Time of exponentiation = 19209
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 153
 Execution Time of exponentiation = 17624
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 196
 Execution Time of exponentiation = 24730
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 239
 Execution Time of exponentiation = 18345
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 175
 Execution Time of exponentiation = 17527
[Hyeseongui-MacBook-Pro:softwareProject hyeseongkim$ java ExecTime
 Execution Time of multipication = 245
 Execution Time of exponentiation = 25319
```

▲ **Fig. 5.** Result of the project

1. a^2을 계산을 계산하는 과정에서 지수승을 사용하는 계산 방법이 $a \times a$ 로 계산하는 방법보다 시간적인 측면에서 덜 효율적이다.

2. 반복적인 수행 결과, 측정값의 편차가 있다.

5. Discussion

이 프로젝트에서 고찰할 점은 다음 세 가지이다.
1. 이 프로젝트에서 수행시간이 '0'인 경우가 나타나지 않은 원인이 무엇인가?
2. 수행시간이 짧아서, nano-second 보다 작다면 어떻게 측정할 것인가?
3. 측정값의 편차가 크다면 어떻게 보정할 것인가?

이 고찰점에서 Windows로 수행한 친구의 결과값이 수행시간이 0인 경우가 나타난 것에 비해, Mac OS로 수행한 본인의 결과값에는 수행시간이 0인 경우가 나타나지 않는 것으로 보아 OS의 차이가 결과에 영향을 미칠 것이라 생각하여 우선 Java와 OS의 관계를 생각해보았다. 하지만 Java 언어의 장점 중 하나가 운영체제에 독립적이라는 점이었다. 자바 응용프로그램은 운영체제나 하드웨어가 아닌 JVM하고만 통신하고 JVM이 자바 응용프로그램으로부터 전달받은 명령을 해당 운영체제가 이해할 수 있도록 변환하여 전달한다. [3] 그렇다면 Java는 운영체제에 독립적이지만 JVM은 운영체제에 종속적이라는 결과를 도출할 수 있다.

따라서, 첫번째 고찰점의 답을 생각해 낼 수 있었다. Oracle에서 제공하는 System class의 Java doc에서 System.nanoTime( ) method의 설명은 다음과 같다.

" Returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds.

…

The same origin is used by all invocations of this method in an instance of a Java virtual machine; other virtual machine instances are likely to use a different origin. "

즉, System.nanoTime( ) method는 JVM을 기준으로 측정하므로 운영체제로부터 시간을 가져온다. 따라서 다른 JVM에서 측정 기준이 다르므로 Mac OS 환경에서 수행한 본인의 결과가 Windows 환경에서 수행한 결과와 달랐다. Windows와 Unix에서 System.nanoTime( ) method의 수행방법의 차이점은, Windows는 CPU에서 읽은 TSC값을 동기화하지만 Unix는 동기화하지 않는다는 것이다. 즉, Windows에서 CPU의 의존성이 더 강하고 CPU간의 차이를 보정해주지 않는다. 따라서 CPU의 클럭 속도에 영향을 받는다. [8]

두번째 고찰점의 답은 실행시간 측정의 여러 method 비교에서 얻을 수 있었다. 실행시간 측정의 method로 보통 System.currentTimeMillis( )와 이 paper에서 쓴 System.nanoTime( ) method가 있다. 여기에 method가 하나 더 있다. JDK 1.5가 도입한 ThreadMXBean 인터페이스에는 getCurrentThreadCpuTime 이라는 method가 있다. 이는 System.nanoTime( )보다 더 작은 실행시간을 산출하는 경향이 있다. 따라서 이를 활용한다면 더 작은 실행시간을 측정할 수 있을 것이다. [5][10]

세번째 고찰점의 해결점은 두가지가 있다. 우선, nanoTime method로 측정할 때 Latency와 Granularity의 문제가 있다. [9] 따라서, static void Thread.sleep(long millis) method를 이용하여 딜레이를 어느정도 보정할 수 있을 것이다. 그 후, 딜레이가 보정된 수행을 반복적으로 하여 평균값을 도출한다. 결과값의 신뢰성을 높이기 위해서는 많은 측정과 통계가 필요하다. [6]

6. References

[1] Oracle Docs, “Class Math”, https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html#method_summary

[2] Oracle Docs, “Class System”, https://docs.oracle.com/javase/7/docs/api/java/lang/System.html

[3] 남궁 성(2008). Java의 정석. 도우출판

[4] 위키백과, 우리 모두의 백과사전, “Vim”, https://ko.wikipedia.org/wiki/Vim

[5] Brent Boyer, “Robust Java benchmarking, Part 1”, https://www.ibm.com/developerworks/library/j-benchmark1/index.html?mhq=Brent%20Boyer&mhsrc=ibmsearch_a

[6] Brent Boyer, “Robust Java benchmarking, Part 2”, https://www.ibm.com/developerworks/java/library/j-benchmark2/index.html

[7] “DK-6440250 : On Windows System.nanoTime() may be 25x slower than System.currentTimeMillis()”, https://bugs.java.com/bugdatabase/view_bug.do?bug_id=6440250

[8] Stanislav Kobylansky (2012), “What is behind System.nanoTime( )?”, https://www.javacodegeeks.com/2012/02/what-is-behind-systemnanotime.html

[9] Aleksey Shipilëv (2014), “Nanotrusting the Nanotime”, https://shipilev.net/blog/2014/nanotrusting-nanotime/

[10] “Performance of synchronize section in Java”, Stack overflow, https://stackoverflow.com/questions/8521819/performance-of-synchronize-section-in-java

7. Evaluation

평가 항목	학생 자체 평가 (리포트 해당 부분 표시 및 간단한 의견)	평가 (빈칸)	점수 (빈칸)
에디터와 명령어를 이용한 실행 - 각 단계별 결과물과 그 의미는? - 요구사항을 만족한 구현? - 충분한 실험? (수행시간이 작은 경우 포함?)	에디터와 명령어를 이용한 실행함 - 자바 실행 모델의 각 단계별 결과물을 사진 첨부 및 과정 설명 - 에디터로 프로그램을 작성하고 cmd 창으로 단계적 실행 - 충분한 실험 사진 첨부 (수행시간이 작은 경우가 나타나지 않아 원인 고찰)		
Eclipse를 이용한 실행	- Eclipse 실행 방법 기재 - Eclipse 실행 사진 첨부		
추가 질문에 대한 의견	- OS에 차이에 따른 결과가 다를 수 있다는 점을 설명함 (수행시간이 작은 경우가 나타나지 않음) - 각 질문에 대해 여러 문헌들을 참고하여 의견을 제시함		
리포트 작성 - 평가 항목에 맞게 리포트 작성? - 모든 파일을 하나의 문서로	- 프로그램의 동작 여부를 알아볼 수 있는 결과 사진 첨부 - 방법에 따른 설명 - 평가자가 보고자 하는 부분 강조(밑줄) - 실제 수행했음을 증명할 수 있는 사진 첨부 - 평가자가 찾기 쉽게 링크 - 모든 파일을 하나의 문서로		
기타 추가 설명 (필요한 경우)	- 추가질문에 프로젝트를 수행하면서 발견한 고찰점을 추가하였다.		
총평/계	평가자 입장에서 자신의 리포트를 살펴보기가 목적 즉, 평가자가 체크하고자하는 사항을 쉽게 찾아볼 수 있도록 리포트가 기술되어있는지 점검		