<JSTORM>

JAVA GUI 실전 프로그래밍 - 1

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

Document Information

Document title:	JAVA GUI 실전 프로그래밍 - 1
Document file name:	자바GUI실전프로그래밍1_장대원.doc
Revision number:	<1.0>
Issued by:	<장대원> dwjang@jkdsoft.co.kr
	정리 : <윤준호> junoyoon@orgio.net
Issue Date:	<2000/8/23 >
Status:	final

Content Information

Audience	마이크로 소프트웨어 독자 자바의 GUI 프로그래밍에 대하여 알고 싶어하는 초보 프로그래머
Abstract	AWT 프로그래밍 - 간단한 AWT 그래픽 에디터 프로그램을 작성함으로써 자바 GUI의 기초를 익힌다.
Reference	
Benchmark information	

Document Approvals

고문	Signature	date
		
지위	Signature	date

.

Revision History

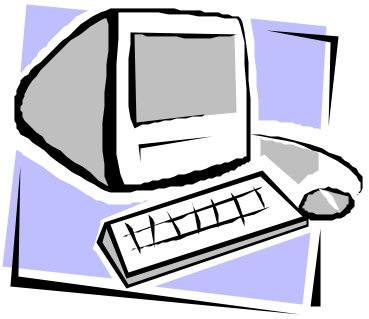
Revision	Date	Author	Description of change

Table of Contents

JAVA GUI 실전 프로그래밍

마소 주니어를 통해 지난 6개월간 독자 여러분들은 초보 프로그래머의 입장에 서서 자바의 개념적인 부분들을 살펴보았습니다. 이제 이번 달부터 6개월간 여러분들은 앞서 배웠던 개념들을 실제 프로그래밍에 적용할 수 있는 능력을 키우는 과정을 필자와 함께 할 것입니다. 어떤 언어를 배운다는 것이 결국은 실제로 프로그램을 만들기 위한 것이라는 것을 생각해볼 때 실전 프로그래밍 능력을 키운다는 것은 매우 중요한 일일 것입니다. 필자의 경험에 의하면 실전 프로그래밍 경험은 거꾸로 이론과 개념에 대한 이해를 탄탄하게 해주는 피드백 성향이 있습니다. 아직 확실하지 않은 개념들은 실전 프로그래밍을 통해 더욱 명확해질 것입니다.

실전 프로그래밍의 중요성에 대해 말씀 드렸는데 실제로 프로그램을 작성하려면 그래도 최소한의 기본 개념은 필요합니다. 1월에서 6월까지 윤경구님 께서 쉽고 친절하게 자바 전반에 걸친 기본 개념들을 잘 정리하여 주셨습니다. 따라서 독자 여러분들이 필자의 글을 이해하는데도 이전의 글들이 유용하게 참조될 것입니다. 그러니 아직 자바의 기초 개념이 전혀 잡혀있지 않다고 느끼시는 분들은 이전의 글들을 통해 기초적인 개념을 먼저 습득하시기 바랍니다. 그러나 기초 개념이 부족한 독자들에게도 필자의 글이 어렵게 느껴지지 않도록 필요한 부분에서는 상세한 해설을 곁들일 생각입니다.



GUI 정복! 초보 탈출의 시작점: AWT와 스윙(I)

첫 회는 독자 여러분들이 부담 없이 글을 읽어나갈 수 있도록 구성하였습니다. 앞으로 계속될 연재를 위한 몇 가지 준비 작업과 초보 프로그래머들에게 필자가 해주고 싶었던 이야기들을 많은 부분을 할애하여 군데군데 실어보았습니다. 이 글을 통해 독자 여러분들이 필자의 글에 빨리 익숙해지기를 기대합니다.

자바를 배우기 위한 필수 문서들

자바에 초보 프로그래머이든 전문가든 여기에 필자가 제시하는 문서들은 필수적으로 갖추고 있어야한다. 이 문서들은 소위 자바의 교과서라고 할 수 있는 것들이다. 다만 교과서는 참고서의 도움이 필요하지만 아래 문서들은 이들만으로도 거의 모든 문제들이 해결 될 수 있을 만큼 자바의 전반적인 부분을 다루고 있는 문서들이다. 필자는 이 문서들을 모두 한 장의 CD에 넣어 보관하고 있습니다. 그리고 필자의 바탕화면에는 이들에 대한 단축 아이콘이 놓여 있습니다. 언제고 필요할 때 즉석에서 문서를 참조하기 위해서죠. 독자 여러분들도 다음 문서들을 반드시 구해서 손쉽고 빠르게 참조할 수 있도록 나름대로의 방법을 강구해 놓기 바랍니다. 구하는 방법은 아래에 필자가 제시하는 공식 웹사이트를 통해서도 가능하며 각 PC 통신의 자료실에서도 쉽게 구할 수 있습니다.

자바 튜토리얼

자바 언어를 탄생시킨 선(Sun)사에서 제공하는 공식 자바 교과서(?)입니다. 자바 언어의 문법과 언어적 특성은 물론이고 JDK(현재 버전 1.2)가 지원하는 거의 대부분의 클래스 라이브러리의 사용법 등이 예제와 그림들이 곁들여진 형태로 아주 자세하게 잘 설명되어 있습니다. 자바에 초보 프로그래머라면 절대 필수적인 문서라 하겠습니다. 이 문서는 책 형태로도 출판되고 있으며 이미 한글판(현재 버전 1.1)이 시중에서 팔리고 있습니다.

URL: http://java.sun.com/docs/books/tutorial/index.html
다운로드: 두 개의 파일을 모두 받아야 합니다.
http://java.sun.com/docs/books/tutorial/download/tutorial.zip
http://java.sun.com/docs/books/tutorial/download/tut-OLDui.zip

자바 API 문서

자바 튜토리얼이 클래스 라이브러리의 사용법을 설명하고 있다면 이 문서는 클래스 라이브러리의 완전한 레퍼런스를 제공하고 있습니다. 즉, 각 클래스가 어떠한 메쏘드나 속성(Attribute)들을 정의하고 있는지 이 문서를 통해 알아 낼 수 있습니다. 따라서 튜토리얼과 병행해서 사용한다면 자바 공부를 하는 사람에게는 좌청룡 우백호를 두는 것이나 마찬가집니다.

URL:
http://java.sun.com/products/jdk/1.2/docs/index.html
위의 사이트는 JDK 1.2의 문서들에 대한 대문 역할을 하는 사이트고 클래스 라이브러리의 실질적 참조를 위해서는 다음 사이트를 통해 직접 참조하는 것이 낫습니다.
http://java.sun.com/products/jdk/1.2/docs/api/index.html
또는
http://java.sun.com/products/jdk/1.2/docs/api/overview -summary.html

다운로드:
http://java.sun.com/products/jdk/1.2/download-docs.html

API 소스

선사가 배포하는 JDK 안에는 자바 클래스 라이브러리 전체(네이티브 코드 제외)의 소스코드가 포함되어 있습니다.(단, 네이트브 코드의 소스는 제외) 정상적으로 JDK를 설치하였다면 JDK의 루트(예를 들어 c:\jdk1.2에 JDK를 설치하였다면 c:\jdk1.2)에 src.jar이라는 파일이 있음을 알 수 있다. 이 파일이 소스 파일인데 이름을 src.zip으로 고친 다음 윈집 등으로 직접 풀 수도 있습니다. 자바의 소스코드는 때로 아주 유용하게 쓰입니다. 클래스 라이브러리에서 뭔가 애매한 부분이 있거나 또는 어떤 메쏘드가 실제로 어떻게 구현되어 있는지 궁굼할 때 바로 이 소스코드를 찾아보면 속 시원히 해결 할 수 있습니다.

예를 들어 다음과 같은 코드가 있다고 해보죠.

```
Button myBtn = new Button("안녕“);
```

독자 여러분은 new Button("안녕“) 이렇게 할 때 내부적으로 어떤 일이 일어나는 지 궁굼할 수 있습니다. 이런 경우 만약 여러분이 java\awt\Button.java 파일을 열어서 읽어본다면 Button 클래스 내부에 String형의 label이라는 변수가 있어서 그 곳에 ”안녕“이라는 문자열이 저장된다는 것을 알 수 있습니다. 자바에 대해 깊이 있게 알고 싶다거나 뭔가를 속 시원히(실재 소스코드를 보는 것보다 속 시원한 방법이 있을까?) 알고 싶다면 자바의 소스코드를 직접 참조할 수 있습니다. 빠른 시일 안에 자바의 전문가가 되고싶은 사람은 소스코드를 살피는데 인색하지 말아야겠습니다. 그러나 역시 초보에게는 어려울 수 있습니다.

자바월드(Java World)

자바월드는 웹 상에서 공짜로 운영되는 자바관련 잡지다. 이곳은 필자 외의 다른 자바 필자들도 항상 빼놓

지 않고 추천하는 유명한 곳이다. 이 곳에 가면 여러 가지 자바 관련 테크닉들(보통은 잘 알려져 있지 않지만 매우 유용한 테크닉들)이나 최근에 이슈가 되는 문제들 또는 자바와 관련된 최근 소식들을 많이 접할 수 있습니다. 특히 이 곳의 어떤 필자들은 아주 인기가 좋아서 팬들이 있을 정돕니다. 필자도 개인적으로 무척 좋아하는 필자(Bill Venners)가 있습니다.

URL:

http://www.javaworld.com

[화면] 자바월드(Java World)의 홈페이지

어떻게 하면 (자바)프로그래밍 공부를 잘 할 수 있을까?

얼마 전 한 전산과 초년생으로부터 어떻게 프로그래밍 공부를 해야할 것인가에 대한 질문 메일을 받은 적이 있습니다. 아마 이 글을 읽는 모든 독자들의 공통된 궁금증이 아닐까하는 생각이 들어 이 곳에 몇 마디 적어보겠습니다.

학구열은 만드는 것이다.

먼저 실력 있는 프로그래머가 되기 위해서는 자신이 하는 공부를 즐길 줄 알아야 합니다. 논어에 이런 대목이 있습니다.

知之者는 不如好之者요 (지지자는 불여호지자요)
好之者는 不如樂之者라 (호지자는 불여락지자라)

"아는 것은 좋아하는 것만 못하고 좋아하는 것은 즐기는 것만 못하다."라는 뜻입니다. 이미 여러분들도 공감하고 계시는 내용일 것입니다. 그러나 공부를 즐기는 경지에 이르기는 참으로 어려운 일입니다. 필자는 흥미를 항상 유지하기 위해 컴퓨터 잡지나 각종 웹사이트를 통해 갖가지 최신 정보들을 가능한 빨리 접하려고 노력하는 편입니다. 마소와 같은 컴퓨터 잡지를 구입하게되면 "편집자 노트"나 그 달의 특집 기사 등을 비롯해서 최신의 동향에 대한 기사들을 꼭 빼지 않고 읽어봅니다. 그리고 자바를 공부하면서부터는 거의 매일 선(Sun)사의 자바 사이트(java.sun.com)를 들러보았습니다. 남들보다 빨리 정보를 접한다는 것, 남

들이 알지 못하는 정보들을 알고 있다는 것, 전문가들이 예상하는 컴퓨팅 업계의 동향을 파악하고 있다는 것은 그 분야에 대한 흥미를 지속적으로 유지시키는 원동력이 됩니다. 사실 이러한 내용들은 특정 분야에 기반 지식이 없더라도 소화가 가능하기 때문에 초보 프로그래머 여러분들도 관심을 갖고 열심히 정보들을 수집하다보면 어느새 학구열에 불타고있는(?) 자신을 발견하게 될 것입니다.

학구열에 대한 조언 말고도 필자가 독자 여러분들에게 이야기해주고 싶은 것들이 몇 가지 있는데 연재 기간 중간 중간에 틈틈이 실어볼 생각입니다. 필자도 여러분들처럼 초보 독자의 입장이었을 때가 있었습니다. 그리고 그 때엔 한 줄의 프로그램 코드보다 먼저 공부한 선배들의 조언 한마디가 아쉬웠던 것을 기억합니다. 모든 독자들에게 완벽한 해답을 제공할 수는 없지만 필자의 경험을 독자 여러분들과 나눌 수 있도록 노력하겠습니다.

GUI! 초보 프로그래머들의 딜레마

자바를 이제 처음 배우는 초보 프로그래머들이 쓸만한 프로그램을 손수 제작해 보려고 할 때 가장 먼저 마주치는 벽이 바로 GUI 부분입니다. 특히 자바로 프로그램을 제작할 때에는 프로그램의 플랫폼 독립성을 지원해야하기 때문에 GUI 설계 시에 고려해야 할 것들이 더 많습니다. 흥미로운 예제를 통해 AWT와 스윙을 쉽고 효율적으로 사용하는 방법을 익히고 더 나아가 멀티 미디어 시대에 걸맞는 깔끔하고 기능적인 GUI 디자인에도 도전해봅시다.

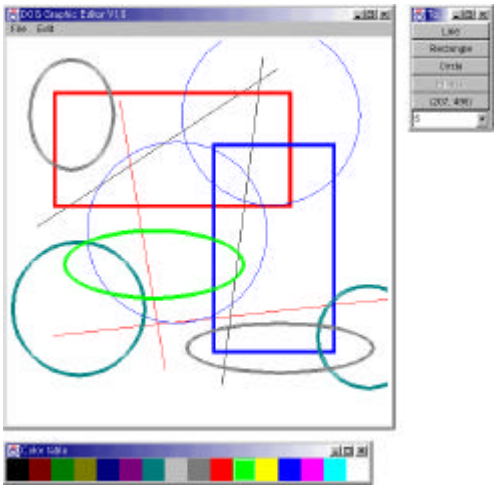
이 글을 이해하기 위해서는 약간의 GUI 지식이 필요합니다. 혹시 마소 주니어 2월 호의 “흥미집중! AWT와 애플릿”을 읽지 않은 독자가 있다면 이 글을 읽기 전에 반드시 한번 살펴보고 오기를 부탁드립니다. 필자가 소스코드 한 줄 한 줄에 담겨있는 기본 개념들을 일일이 재탕(?)할 수는 없기 때문입니다. 물론 그에 상응하는 기반 지식을 이미 가지고 계신 분은 문제없겠지만 GUI에 대한 기초 개념이 전혀 없으신 분은 글을 이해하는데 어려움을 있을 수 있습니다.

이번 달과 다음 달 이렇게 두 달 동안 독자 여러분은 자바 프로그램의 GUI 부분을 프로그래밍 하는 방법을 익히게 될 것입니다. 이를 위해 필자는 두 개의 프로그램을 준비했습니다. 그 중 하나는 이번 회에 보게 될 것이고 다른 하나는 다음 회에 보게될 것입니다. 그런데 사실은 두 프로그램이 거의 같은 프로그램입니다. 종류도 같고 그 기능도 거의 비슷할 것입니다. 필자가 GEditor라고 이름 붙인 첫 번째 프로그램은 필자가 자바 초보시절에(정확히는 초보의 후반이라고 해야겠군요. 아무튼 필자 개인적으로는 초보시절이라고 생각되는 때에) 그때까지 배웠던 자바 언어를 실습한다는 의미로 작성해 본 그래픽 에디터입니다. 정확히 장르를 구분한다면 캐드라고 해야 맞을 것입니다. 왜냐하면 이 프로그램이 벡터 그래픽만을 지원하기 때문입니다. [화면 1]에서 프로그램의 실행 모습을 볼 수 있습니다.

필자가 굳이 초보 시절에 만들었던 프로그램을 내어놓는 이유는 먼저 초보의 입장에서 프로그램을 바라보고자 하는 의도입니다. 좀 엉성한 프로그램을 보면서 그 프로그램의

문제점은 뭐고 또 그러한 문제점을 어떻게 개선할 수 있는지 살펴보는 것은 독자 여러분들에게 큰 도움이 되리라 생각됩니다. 또 다른 이유는 프로그램의 규모가 그리 크지 않다는 것입니다. 즉, 독자 여러분들이 충분히 소화할 수 있는 프로그램이라 생각되어 이 프로그램을 선택하게 된 것입니다. 독자 여러분들은 프로그램의 GUI 부분에 대해 중점적으로 관심을 갖고 프로그램 분석을 따라오시면 되겠습니다. 그럼 프로그램에 대해 먼저 간략히 알아볼까요?

[화면 1]의 메인 창(Main Frame)을 보면 창안에 여러 도형들이 다양한 모양과 색상 그리고 여러 가지의 외곽선 두께로 그려져 있습니다. 오른쪽의 툴바 창은 도형의 종류와 외곽선의 두께를 선택할 수 있는 곳이고 아래쪽의 컬러 테이블은 도형의 색상을 선택하는 곳입니다. 윈도우98의 그림판 프로그램을 연상하시면 되겠습니다. 단 이 프로그램은 벡터 그래픽용 프로그램이라는 것이 다른 점이지요. 프로그램이 시작되면 이 세 개의 창이 동시에 화면에 나타나고 사용자는 원하는 도형과 색상을 선택한 다음 메인 창의 캔버스에서 마우스를 드래그 하는 것으로 여러 도형을 그릴 수 있습니다. 그럼 실제 프로그램이 어떤 과정을 통해 구현되었는지 살펴보죠.

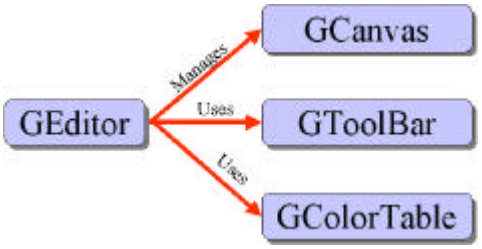


[화면 1] GEditor의 실행 화면

프로그램의 구성 요소들

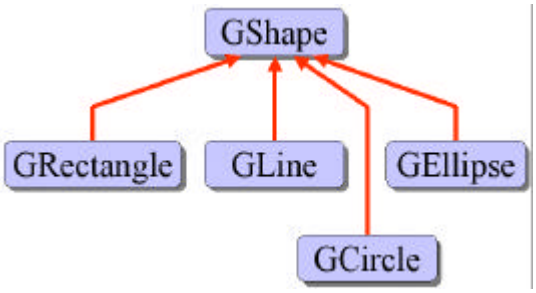
어떤 프로그램이든 프로그램을 작성하기 이전에 먼저 프로그램들이 어떤 구성 요소로 나뉘어질 수 있는지를 생각해 보아야 합니다. 객체지향 프로그래밍 언어를 배우는 사람으로서 너무나 당연한 얘기죠. GEditor를 구성하는 구성 요소(Entity)들은 [그림 1]에서 볼 수 있는 것처럼 GEditor, GCanvas, GToolBar 그리고 GColorTable 등이 있습니다. GCanvas는 도형이 그려질 캔버스를 나타내고 GToolBar는 도형의 모양과 외곽선의 두께를 선택할 수 있는 툴바를 나타내며 GColorTable은 도형의 색상을 선택할 수 있는 컬러 테이블을 나타냅니다. 마지막으로 GEditor는 툴바와 컬러테이블에 현재 선택되어진 도형의 각종

속성 값(모양, 외곽선의 두께, 색상)을 이용하여 캔버스에 도형을 그리는 작업을 수행하는 에디터를 나타냅니다. 독자들도 어떤 프로그램을 만들고자할 때에는 필자처럼 먼저 어떤 구성 요소들이 필요한지 결정한 다음 이들 구성 요소들의 맡은바 임무가 무엇인지를 명확히 정의하는 습관을 들이시기 바랍니다. ([박스 기사 참조]: 분석! 설계! 반드시 문서화하자.)



[그림 1] GEditor의 각 구성 요소들간의 관계

다음으로 정의해야 할 것은 그래픽 객체들입니다. [그림 2]에 그래픽 객체들의 상속 관계를 나타내어 보았습니다. 먼저 모든 도형들의 공통 집합을 추출해서 GShape이라는 클래스를 정의한 다음 구체적인 도형들, 예를 들어 직선, 사각형, 원, 타원 등은 GShape을 상속(extends)하여 도형 클래스를 구성할 수 있을 것입니다.



[그림 2] 그래픽 객체들의 상속 계층도

필자가 설계한 구조도가 썩 훌륭한 편은 못 됩니다. 독자 여러분들은 다음 회에 좀 더 나은 버전의 설계도를 볼 수 있을 것입니다. 그럼 이제부터 각 구성 요소들의 구체적인 구현 방법들을 살펴보겠습니다.

툴바(GToolBar) 클래스

맨 먼저 툴바를 나타내는 GToolBar 클래스를 살펴보겠습니다.([리스트 1]) 앞에서도 말했었듯이 GToolBar 클래스의 임무는 도형의 종류와 외곽선의 두께를 선택하는데 그 목적이 있습니다. 따라서 다음과 같은 두 개의 메쏘드가 그 핵심이라 하겠습니다. 이 두 개의 메쏘드는 GEditor에 의해 호출될 것입니다.

```
getSelectedShape()
getSelectedThickness()
```

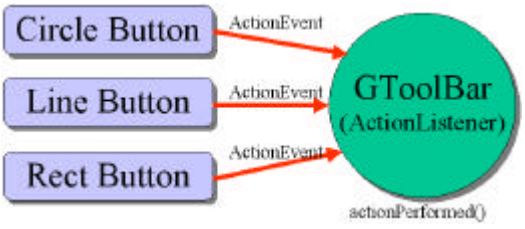
소스코드에 주석이 많아서 이 클래스를 이해하는 데 특별한 어려움은 없을 것으로 생각됩니다. 다만 자바에 초보자라면 이벤트 처리 부분이 어려울 수 있을 것 같으니 툴바에 쓰인 이벤트 처리에 대해 간략히 알아보겠습니다.

이벤트 처리는 GUI 프로그래밍에 있어서 가장 중요한 부분이라고 할 수 있습니다. 따라서 이벤트 처리에 대한 이해는 독자 여러분들이 GUI를 정복하기 위해서 가장 중요시해야 할 부분이기도 합니다. 이벤트 처리에 대한 구체적인 이야기는 "마소 Jr. 2월 호의 16 페이지"를 참조하시기 바랍니다.

[리스트 1]에서 [2번]은 도형 버튼들에 Action 이벤트 처리기를 등록하는 곳입니다.(버튼이 눌릴 때에는 Action 이벤트가 발생한다) 그런데 특이한 점은 addActionListener()의 매개변수로 툴바 자신(this)을 넘겨줍니다. 즉 버튼이 눌리는 것을 툴바 자체에서 처리하겠다는 것입니다. addActionListener()의 매개변수로 넘어가는 객체는 ActionListener 인터페이스를 구현하는(implements)하는 객체여야 하는 것은 독자 여러분들도 아시리라 믿습니다. 이런 기반 사항을 모르시는 분은 앞서 말씀드린 것처럼 "마소 Jr."의 이벤트 부분을 꼭 살펴보시기 바랍니다. 후일이라도 지면 여유가 좀 생기면 자바의 이벤트에 대해 심도 있게 다뤄볼 생각입니다만 우선은 간략하게라도 이벤트에 대해 알아두시기 바랍니다.

아무튼 소스코드의 [2번]에서처럼 툴바 자신이 addActionListener()의 매개변수로 넘어가하기 때문에 GToolBar 클래스는 ActionListener 인터페이스를 구현해야 합니다.([1번]) ActionListener 인터페이스를 구현한다는 것은 곧 actionPerformed() 메쏘드를 만들어 놓는다는 것과 같습니다. actionPerformed() 메쏘드는 버튼이 눌렸을 때 버튼으로부터 호출되는 메쏘드이기 때문에([그림 3] 참조) 도형 버튼이 눌렸을 때 버튼 눌림에 대한 처리를 이 곳에서 수행하면 됩니다. 그럼 actionPerformed() 메쏘드에 대해서도 간략히 알아볼까요?

actionPerformed() 메쏘드에 매개변수로 들어오는 ActionEvent 객체에는 여러 가지 정보가 포함되어 있습니다. 예를 들어 Action 이벤트가 발생한 버튼의 레이블 문자열을 얻어낼 수도 있습니다. 우리의 툴바 클래스에서는 바로 이 문자열을 이용하여 여러 버튼들(라인 버튼, 사각형 버튼 등등...) 중에 어떤 버튼이 눌렸는지를 판단하게 됩니다. ActionEvent의 getActionCommand() 메쏘드를 이용하면 눌린 버튼의 레이블을 얻어올 수 있습니다. (실제로는 setActionCommand()로 명령 문자열을 명확히 지정하지 않았을 때만 버튼의 레이블을 얻어오게 된다. 그러나 신경 쓸 필요는 없다.) 이 정도로 툴바에 대한 이야기는 끝내겠습니다.



[그림 3] 툴바에서의 Action 이벤트

[표 1] 자주 쓰이는 이벤트 리스너와 그 메쏘드들

WindowListener	public void windowOpened(WindowEvent e)
	public void windowClosing(WindowEvent e)
	public void windowClosed(WindowEvent e)
	public void windowIconified(WindowEvent e)
	public void windowDeiconified(WindowEvent e)
	public void windowActivated(WindowEvent e)
	public void windowDeactivated(WindowEvent e)
MouseMotionListener	public void mouseDragged(MouseEvent e)
	public void mouseMoved(MouseEvent e)
MouseListener	public void mouseClicked(MouseEvent e)
	public void mousePressed(MouseEvent e)
	public void mouseReleased(MouseEvent e)
	public void mouseEntered(MouseEvent e)
	public void mouseExited(MouseEvent e)
KeyListener	public void keyTyped(KeyEvent e)
	public void keyPressed(KeyEvent e)
	public void keyReleased(KeyEvent e)
ActionListener	public void actionPerformed(ActionEvent e)

[리스트 1] GToolBar.java

```
import java.awt.*;
import java.awt.event.*;

////////////////////////////////////////////////////////////////////
class GToolBar extends Frame implements ActionListener {     <--- [1번]
   // 도형 버튼의 레이블로 사용될 문자열들
   static final String[] labels = {
      "Line", "Rectangle", "Circle", "Ellipse"
   };

   Button[] m_buttons;        // 도형 버튼들
   Button m_posButton;        // (마우스)포인터의 위치 표시 버튼
   Choice m_lineThickness;    // 외곽선 두께 선택
   int m_selectedShape;       // 현재 선택되어 있는 도형의 종류

   // ----------------------------------------------------------------
   public GToolBar() {
      super("Tool Bar");

      // 각종 버튼들과 선택상자를 생성한다.
      m_buttons = new Button[GToolBar.labels.length];
      m_posButton = new Button("");
      m_lineThickness = new Choice();

      // GridLayout으로 버튼들을 배열한다. GridLayout의 생성자에는 세로
      // 방향으로의 격자수와 가로방향으로의 격자 수가 매개변수로 넘어간다.
      // 이때 격자수에 0을 넣으면 그 방향으로는 얼마든지 컴포넌트를 넣을
      // 수 있다는 것을 의미한다. 따라서 아래의 "new GridLayout(0, 1)"는
      // 세로방향으로 툴바를 여러 개 배열하겠다는 의미다.
      setLayout(new GridLayout(0, 1));
      for (int i=0; i<GToolBar.labels.length; i++) {
         m_buttons[i] = new Button(labels[i]);
         add(m_buttons[i]);
         m_buttons[i].addActionListener(this);                    <--- [2번]
      }
      add(m_posButton);

      // 외곽선의 두께를 선택할 수 있는 선택 상자를 초기화 한다.
      for (int i=1; i<=10; i++) {
```

```
            m_lineThickness.addItem(""+i);
        }
        add(m_lineThickness);
        pack();
        setVisible(true);
        // 초기 선택 도형을 설정한다.
        m_selectedShape = GShape.line;
        m_buttons[0].setEnabled(false);
    }
    // ---------------------------------------------------------------
    // 도형 버튼의 눌림을 처리한다.
    public void actionPerformed(ActionEvent e) {
        // 버튼의 레이블을 얻어온다.
        String command = e.getActionCommand();
        // 현재 눌려진 버튼만 비활성화 시키고 나머지는 모두
        // 활성화 시킨다. 이때 버튼이 비 활성화되어 있는 것은
        // 해당 도형이 선택되어 있다는 의미다.
        for (int i=0; i<GToolBar.labels.length; i++)
            if (command.equals(labels[i])) {
                m_selectedShape = GShape.line+i;
                m_buttons[i].setEnabled(false);
            }
            else
                m_buttons[i].setEnabled(true);
    }
    // ---------------------------------------------------------------
    // (마우스)포인터의 위치를 표시한다.
    public void printLocation(Point location) {
        m_posButton.setLabel("("+location.x+", "+location.y+")");
    }
    // ---------------------------------------------------------------
    // 툴바에 현재 선택되어 있는 도형의 종류를 얻어낸다.
    public int getSelectedShape() {
        return m_selectedShape;
    }
    // ---------------------------------------------------------------
    // 툴바에 현재 선택되어 있는 외곽선 두께를 얻어낸다.
    public int getSelectedThickness() {
        return m_lineThickness.getSelectedIndex()+1;
    }
}
```

컬러 테이블(GColorTable) 클래스

이번엔 [리스트 2]의 컬러테이블을 살펴볼 차례입니다. 컬러테이블에서는 현재 선택된 색상을 가져오는 것이 목적이므로 컬러테이블의 실제 사용자인 GEditor의 입장에서 보면 다음 메쏘드가 GColorTable에서 가장 중요한 메쏘드라고 할 수 있겠습니다.

```
public Color getSelectedColor()
```

(항상 그 객체의 외부에서 그 객체에게 요구하는 기능이 무엇인지를 먼저 결정해야합니다. 그것이 각 객체들 간의 통신 수단이기 때문이지요. 이러한 기능을 구현하기 위한 방법은 차후에 생각할 일입니다.) 소스코드를 끝까지 보고 이 메쏘드가 어떤 방식으로 구현되어 있는지 살펴보기 바랍니다.

컬러테이블에서 눈여겨볼 것은 Canvas 클래스를 오버라이드(override) 하는 방법입니다. [리스트 2]에 그 방법이 나와 있다. 창안에 무엇인가를 그려야할 필요가 있을 때는 Canvas 클래스 상속한 다음 paint() 메쏘드를 오버라이드하여 이곳에 그리고자 하는 내용을 그리면 된다. ([1번], [2번]) (각종 윈도우 컴포넌트들의 용도를 알고싶으면 "마소 Jr. 2월 호의 14페이지"를 참조하세요.) paint() 메쏘드는 창의 내용물이 그려져야 할 필요가 있을 때 자동으로 호출됩니다. 예를 들어 다른 윈도우가 창을 가렸다가 다시 사라졌을 경우 paint() 메쏘드가 호출됩니다.

[리스트 2]에는 update()라는 메쏘드도 보이는데 이 메쏘드는 paint()와 비슷한 작업을 하는 메쏘드입니다. 사실 화면에 무엇인가가 그려져야 할 상황이 되면 paint() 메쏘드가 직접 호출되는 것이 아니라 update() 메쏘드가 먼저 호출되는 것입니다. 그런 다음 update() 메쏘드가 paint() 메쏘드를 내부에서 호출하는 것이지요. update() 메쏘드는 paint() 메쏘드를 호출하는 것 이외 몇 가지 일을 추가로 수행합니다. 먼저 현재 창을 창에 지정된 배경 색으로 창을 몽땅 채우는 일을 합니다. 즉 화면을 깨끗이 지우는 일을 하는 것이지요. 그런 다음 현재 창에 할당된 Graphics 객체의 색상을 현재 창의 전경 색으로 설정합니다. 그리고 마지막으로 paint() 메쏘드를 호출하는 것이지요. 따라서 update() 메쏘드를 오버라이드하면 경우에 따라서는 배경 무늬를 사용자가 직접 그리거나 또는 배경이 지워지는 단계를 없앰으로써 출력 속도를 향상할 수 있습니다.

컬러 테이블 역시 주석이 자세하기 때문에 큰 어려움 없이 프로그램을 이해할 수 있으리라 생각된다. 혹시 컬러테이블에서 처리하는 마우스 이벤트에 대해 어려움이 있는 독자들은 앞서 필자가 알려주었던 이전 기사를 참조하시기 바랍니다.

[리스트 2] GColorTable.java

```
import java.awt.*;
import java.awt.event.*;

////////////////////////////////////////////////////////////////////////
// 색상을 선택할 수 있는 테이블이다.
// 색상 개수만큼의 셀들로 이루어진 테이블이다.
// getSelectedColor()로 현재 테이블에 선택된 색상을 얻는다.
class GColorTable extends Canvas {                                    <-- [1번]

  // 테이블의 크기
  protected static final Dimension size = new Dimension(16,1);

  // 테이블에 포함될 색상들
  protected static final Color[] colors = {
    new Color(  0, 0  ,  0),
    new Color(128, 0  ,  0),
    new Color(  0, 128,  0),
    new Color(128, 128,  0),
    new Color(  0,  0, 128),
    new Color(128,  0, 128),
    new Color(  0, 128, 128),
    new Color(192, 192, 192),
    new Color(128, 128, 128),
    new Color(255,  0,  0),
    new Color(  0, 255,  0),
    new Color(255, 255,  0),
    new Color(  0,  0, 255),
    new Color(255,  0, 255),
    new Color(  0, 255, 255),
    new Color(255, 255, 255)
  };

  // ---------------------------------------------------------------
  protected Frame m_frame;          // 테이블을 담을 프레임
  protected int m_cellSize = 30;    // 색상 셀의 크기
  protected Color m_selectedColor;  // 현재 선택되어 있는 색상

  // ---------------------------------------------------------------
  // 생성자 :
```

```
public GColorTable() {
   // 프레임을 초기화시키고 컬러테이블을 프레임이 담는다.
    m_frame = new Frame("Color table");
   setSize(m_cellSize*size.width, m_cellSize*size.height);
    m_frame.add(this);
    m_frame.pack();
    m_frame.show();
    m_selectedColor = colors[0];

   // 마우스가 클릭된 셀이 선택되도록한다.
    addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
       Rectangle rectangle = new Rectangle();
       for (int y=0; y<size.height; y++) {
          for (int x=0; x<size.width; x++) {
             // 각 셀별로 사각형을 설정한 다음 해당 셀에
             // 마우스가 클릭되었는지 살핀다.
              rectangle.setBounds(x*m_cellSize, y*m_cellSize,
                    m_cellSize, m_cellSize);
              if (rectangle.contains(e.getPoint()))
                 m_selectedColor = colors[y*size.height+x];
          }
       }
       repaint();
   }});
}
// ---------------------------------------------------------------
// 현재 선택된 색상을 얻는다.
public Color getSelectedColor() {
   return m_selectedColor;
}
// ---------------------------------------------------------------
// 테이블의 위치를 조절한다.
public void setLocation(Point location) {
   m_frame.setLocation(location);
}
// ---------------------------------------------------------------
public void update(Graphics g) {
   paint(g);
}
// ---------------------------------------------------------------
```

```
    public void paint(Graphics g) {                          <--- [2번]
        drawTable(g);
    }
    // ---------------------------------------------------------------
    // 색상 테이블을 그린다.
    protected void drawTable(Graphics g) {
        for (int y=0; y<size.height; y++) {
            for (int x=0; x<size.width; x++) {
                g.setColor(colors[y*size.height+x]);
                g.fillRect(x*m_cellSize, y*m_cellSize,
                        m_cellSize, m_cellSize);
                // 현재 선택된 셀에 선택 사각형을 그린다.
                if (m_selectedColor==colors[y*size.height+x]) {
                    g.setXORMode(Color.white);
                    g.drawRect(x*m_cellSize+1, y*m_cellSize+1,
                            m_cellSize-2, m_cellSize-2);
                    g.setPaintMode();
                }
            }
        }
    }
}
```

캔버스(GCanvas) 클래스

캔버스는 도형이 그려지는 부분입니다. 독자들도 예상할 수 있다시피 우리의 캔버스 클래스(GCanvas)는 java.awt의 Canvas 클래스를 상속(extends)하고 있습니다. 말 그대로 그림이 그려져야 할 곳이기 때문입니다. 아직 우리는 도형 클래스를 살펴보지 못했습니다만 도형 객체들은 다음에 살펴볼 GEditor 객체에 저장되어 있습니다. GCanvas 객체는 GEditor 객체에게서 도형 객체들을 얻어와 그것들을 캔버스에 그리게 됩니다. [2번]이라고 된 곳을 보면 GEditor 객체(m_editor)에 저장되어 있는 도형 객체들(m_shapes)을 가져와 캔버스에 그리는 것을 알 수 있습니다.

```
((GShape)m_editor.m_shapes.elementAt(i)).draw(m_offG);
```

캔버스 안에 도형 객체들이 존재한다면 훨씬 빨리 그리고 쉽게 그림을 그릴 수 있을 것 같은데 왜 GEditor 객체에 도형을 저장하는 것일까요? 이것은 프로그램에 유연성과 확장성을 부여하기 위해서입니다. 예를 들어볼까요? 똑같은 그림을 두 개의 캔버스에 그릴 일이 있다고 가정해보세요. 만약 각각의 캔버스가 도형 객체를 소유하고 있었다면 똑같은 도형이 불필요하게 중복으로 저장되어 있을 것입니다. 더구나 사용자가 어느 하나의 캔버스에 또 다른 도형을 그렸다면 두 캔버스에 모두 도형을 추가해주어야 하는 부담도 있습니다. 캔버스와 캔버스에 그려질 데이터인 도형들을 따로 분리하는 이유는 이 외에도 몇 가지가 더 있습니다.

갑자기 필자가 이런 얘기를 꺼내는 것은 다음 회에 소개할 스윙의 기본 구조인 MVC(Model View Controller) 구조에 대한 느낌을 독자 여러분들에게 복선으로 제공하려는 의도입니다. 일단 여기서는 데이터와 데이터를 표현하는 객체는 분리하는 것이 좀 더 나은 방법이라는 것만 느낌으로 간직하고 계시기 바랍니다. 전혀 대비 없이 다음 호에서 MVC 구조에 대한 이야기를 들으면 갑자기 뇌에 충격(?)이 올지도 모릅니다.

사용자가 그림을 그리려면 캔버스 위에서 마우스를 누르거나 드래그하게 됩니다. 따라서 캔버스의 중요한 기능 중에 하나는 마우스 처리에 관한 것이지요. 그런데 우리의 캔버스는 마우스를 자신이 직접 처리하지 않고 자신을 관리하는 GEditor 객체에게 그것을 위임합니다. ([리스트 3]의 [1번]) 필자 생각에 좀 개선이 필요한 부분으로 여겨집니다. 다음 호에 개선된 내용을 소개해 드리겠습니다. 아무튼 이렇게 되면 마우스 처리에 관해서는 GEditor 클래스에서 다시 살펴 봐야할 것 같습니다.

[리스트 3]의 paint() 메쏘드 부분을 보면 이상한 코드들이 눈에 띌 것입니다. 우리의 캔버스는 깔끔한 화면 출력을 위해 더블 버퍼링이라는 것을 사용하기 때문입니다. 더블 버퍼링에 관한 내용은 다음 호의 스윙을 배우면서 자동으로 해결될 것이기 때문에 여기서는 그 개념에 대해서만 간략히 알아보겠습니다. 더블 버퍼링은 화면 깜박임을 없애기 위해 도입된 일종의 꼼수(?)라고 볼 수 있습니다. 화면 깜박임이 생기는 이유는 창안에 무엇인

가를 그리는 동안 화면이 갱신되기 때문입니다.(모니터는 항상 매우 빠른 속도로 화면을 갱신하고 있다.) 모두 그려지기 전에 화면이 바뀌기 때문에 그림의 일부는 전 화면에 없다가 나중 화면에 갑자기 나타나는 것이죠. 사용자 입장에서 보면 화면이 깜박이는 것으로 보이게 됩니다. 더블 버퍼링은 이러한 문제를 해결하기 위한 방법으로 화면의 이면에서 그림을 미리 모두 그려 놓은 다음 다 그려진 그림을 한꺼번에 화면에 표시하는 것입니다. 따라서 화면 자체를 위한 버퍼와 이면 작업을 위한 버퍼(Off Screen) 이렇게 두 개의 버퍼가 필요한 것이죠. AWT로 애니메이션을 작성하신다면 더블 버퍼링은 필수일 것입니다. 더블 버퍼링에 관한 이야기는 이 정도로만 해두겠습니다.

[리스트 3] GCanvas.java

```
import java.awt.*;

////////////////////////////////////////////////////////////////////////
// 그래픽 에디터의 캔버스 부분이다. 즉 실제 그림이 그려지는 곳이다.
// MFC의 Document/View 구조에 비유할때 GEditor가 Document 부분이라면
// 이곳은 View 부분이다. 따라서 그려져야 할 객체가 있을경우 GEditor
// 객체에서 data를 참조하여 화면에 출력한다.
// 더블 버퍼링을 사용하였으며 GCanvas 에 발생된 모든 마우스 이벤트를
// GEditor 객체가 처리하도록 하였다.
class GCanvas extends Canvas {

   // 캔버스의 디폴트 크기
   protected static final Dimension sizeDefault
         = new Dimension(500, 500);

   // GEditor 객체 (Document)
   protected GEditor m_editor;

   // double buffering을 위한 변수들
   protected Graphics m_offG;      // off screen Graphics
   protected Image m_offS;         // off screen

   // ----------------------------------------------------------------
   // 생성자 : GEditor 객체를 매개변수로 받는다.
   public GCanvas(GEditor editor) {
      m_editor = editor;

      setSize(sizeDefault);
      addMouseListener(m_editor);                    <--- [1번]
```

```
        addMouseMotionListener(m_editor);                    <--- [1번]
    }

    // ----------------------------------------------------------------
    // 전 화면 갱신때 불려진다. 이때 off screen 이미지를 다시 설정한다
    // 캔버스의 크기가 변할때 능동적으로 screen 이미지도 변화시킨다.
    public void invalidate() {
        super.invalidate();
        m_offS = null;
    }

    // ----------------------------------------------------------------
    public void update(Graphics g) {
        paint(g);
    }

    // ----------------------------------------------------------------
    // 더블 버퍼링을 이용하여 화면에 GShpe 객체들을 그린다.
    // 이때 GShpe 객체들은 GEditor 객체로부터 참조된다.
    public void paint(Graphics g) {

        // 더블 버퍼링용 이미지와 Graphics를 초기화 한다.
        if (m_offS==null)
            m_offS = createImage(getSize().width, getSize().height);
        m_offG = m_offS.getGraphics();
        m_offG.setClip(0, 0, getSize().width, getSize().height);

        // 실제 GShape 객체들이 그려진다.
        int shapeCount = m_editor.m_shapes.size();
        for (int i=0; i<shapeCount; I++)
            ((GShape)m_editor.m_shapes.elementAt(i))          <--- [2번]
                .draw(m_offG);

        g.drawImage(m_offS, 0, 0, null);
        m_offG.dispose();
    }
}
```

에디터(GEditor) 클래스

[리스트 4]에 있는 에디터 클래스는 앞서 살펴보았던 툴바, 컬러 테이블, 캔버스 클래스를 관리하는 메니져 클래스입니다. 여기에 더하여 도형 객체들을 저장하는 저장소이기도 하지요.([1번]) 프로그램에서 가장 중요한 클래스라고 생각할 수 있겠습니다. 그러나 구조가 복잡하지 않고 주석이 많으므로 특별한 설명은 생략하겠습니다.

한가지 살펴 볼만한 것은 [2번]이라고 되어 있는 곳입니다. paint() 메쏘드에는 Graphics 객체가 매개변수로 넘어오기 때문에 그 객체로 그리는 작업을 하면 됩니다. 그러나 paint() 메쏘드 이외의 곳에서도 그리는 작업이 필요할 수 있습니다. 이때는 그림이 그려질 컴포넌트의 getGraphics() 메쏘드를 통해 Graphics 객체를 얻어오면 됩니다. 각 컴포넌트마다 자신에게 할당된 Graphics 객체가 하나씩 존재하는데 그것을 얻어오면 아무 데서나 자유롭게 그래픽 작업을 할 수 있습니다.

[리스트 4] GEitor.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.io.*;

////////////////////////////////////////////////////////////////////////
// 프로그램의 Main 클래스로서 GCanvas, GToolbar, GColorTable,
// GStatusbar등을 직접 관리한다.
// 특히 캔버스에 그려질 Shape 객체들을 Vector로 관리하며
// Canvas에 발생한 마우스 이벤트들을 처리한다.

class GEditor implements MouseListener, MouseMotionListener {
    ...
    protected GCanvas m_canvas;         // 캔버스
    protected GToolBar m_toolbar;       // 툴바
    protected GColorTable m_colorTable; // 컬러 테이블
    protected GShape m_currShape;       // 현재 선택된 Shpe 객체
    protected int m_shapeKind;          // 현재 선택된 Shpe의 종류
    protected Vector m_shapes;          // Shape 객체들을 저장  <--- [1 번]

    // ------------------------------------------------------------
    // 생성자: Editor가 관리하는 각종 객체들을 생성하고 초기화 시킨다.
    public GEditor() {
```

```
    ...
    m_canvas = new GCanvas(this);
    m_toolbar = new GToolBar();
    m_colorTable = new GColorTable();
    m_shapes = new Vector(1024, 1024);
    ...
}

// ----------------------------------------------------------------
// 캔버스에 발생한 마우스 눌림 이벤트를 처리한다.
// 툴바에서 현재 선택된 종류의 Shape객체를 생성해서
// GColorTable에 선택된 색상등으로 객체를 초기화 시킨다.
public void mousePressed(MouseEvent e) {
    // 툴바로부터 현재 선택되어 있는 도형의 종류를 얻어온다.
    m_shapeKind = m_toolbar.getSelectedShape();

    switch (m_shapeKind) {
    case GShape.line:
    default:
        m_currShape = new GLine(e.getPoint());
        break;
    case GShape.rectangle:
        m_currShape = new GRectangle(e.getPoint());
        break;
    case GShape.circle:
        m_currShape = new GCircle(e.getPoint());
        // 원의 중심에 조그만 사각형을 표시한다.
        m_canvas.getGraphics().drawRect(
                e.getX()-1, e.getY()-1, 2, 2);
        break;
    case GShape.ellipse:
        m_currShape = new GEllipse(e.getPoint());
        break;
    }

    m_currShape.setColor(m_colorTable.getSelectedColor());
    m_currShape.setThickness(m_toolbar.getSelectedThickness());
}

// ----------------------------------------------------------------
// 캔버스에 발생한 마우스 놓기(?) 이벤트를 처리한다.
```

```
// 벡터에 현재 Shape 객체를 저장하고 화면을 갱신한다.
public void mouseReleased(MouseEvent e) {
    if (m_currShape==null)
        return;

    m_shapes.addElement(m_currShape);
    m_currShape = null;
    m_canvas.repaint();
}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}

// ----------------------------------------------------------------
// 캔버스에 발생한 마우스 드래그 이벤트를 처리한다.
// 드래그하는 마우스 위치에 따라 Shape 모양을 연속
//갱신하고, 현재 마우스 위치를 화면에 표시한다.
public void mouseDragged(MouseEvent e) {
    if (m_currShape==null)
        return;

    Graphics g = m_canvas.getGraphics();                    <--- [2번]

    // 현재 모양을 지우고 새로운 위치에 다시 그린다.
    m_currShape.erase(g);
    m_currShape.setNextLocation(e.getPoint());
    m_currShape.draw(g);
    g.dispose();
}

...

// ----------------------------------------------------------------
// 메뉴 처리하는곳
public void setMenu() {
    MenuBar menubar = new MenuBar();
    m_frame.setMenuBar(menubar);
    Menu file = new Menu("File");
    menubar.add(file);
    Menu edit = new Menu("Edit");
    menubar.add(edit);
```

```
        MenuItem load, save, exit;
        file.add(load = new MenuItem("Load",
                new MenuShortcut(KeyEvent.VK_L)));
        file.add(save = new MenuItem("Save",
                new MenuShortcut(KeyEvent.VK_S)));
        file.addSeparator();
        file.add(exit = new MenuItem("Exit",
                new MenuShortcut(KeyEvent.VK_X)));
        MenuItem clear;
        …
    }
    // ---------------------------------------------------------------
    // 파일로 부터 Shape들을 로드한다.
    public void loadShapes() {
        …
    }
    // ---------------------------------------------------------------
    // Shape들을 파일에 저장한다.
    public void saveShapes() {
        …
    }
    // ---------------------------------------------------------------
    // 에디터를 종료한다.
    protected void exitEditor() {
        System.exit(1);
    }
    // ---------------------------------------------------------------
    public static void main(String[] args) {
        new GEditor();
    }
}
```

지면 관계상 도형 클래스에 대한 이야기는 다음 회로 미루겠습니다. 대신 이 달의 디스켓으로 GEditor 프로그램의 전체 소스가 제공되니 한번씩 살펴보기 바랍니다. 글의 앞부분에서 도형 클래스들이 어떤 식으로 이루어졌는지 설명 드렸기 때문에 소스만으로도 충분히 이해할 수 있으리라 생각됩니다.

분석! 설계! 반드시 문서화하자.

더 정확히 말한다면 분석 결과물과 설계 결과물을 문서화(Documentation) 하라는 말입니다. 아마 분석, 설계라는 용어를 처음 들어본 독자도 있을 것입니다.

넓은 의미에서 분석이란 자신이 해결해야할 문제 영역(Problem Domain)을 정확하게 이해하기 위해 필요한 모든 것이라 할 수 있습니다. 예를 들어 캐드 프로그램을 개발할 생각이라면 캐드 프로그램이 어디에 어떻게 쓰이는가를 먼저 정확히 이해할 수 있도록 캐드 프로그램이 지원해야할 기능들(Requirements)이나 그러한 기능들이 실제로 어떻게 사용될 수 있는지(Use Cases) 등을 명백히(문서나 그림으로써) 정의하는 것이 분석입니다.

분석과 함께 반드시 거쳐 가야할 단계로서 설계 과정도 아주 중요한 과정입니다. 일반적으로 분석 과정은 문제 영역이 "무엇(What)"인가에 집중하는데 반해 설계 과정은 문제를 해결하기 위해 어떠한 방법을 사용할 것인가에 관심을 기울입니다. 즉 "어떻게(How)" 문제를 해결할 것인가에 집중하는 것이죠. 설계 과정에서 가장 중요한 부분은 소프트웨어 시스템을 구성하는 각 구성 요소(보통은 객체나 클래스를 말합니다.)들에게 자신들의 임무를 할당(Assigning Responsibilities)하는 과정입니다. 어떻게 임무를 할당해야 구성 요소들 간에 유기적인 상호 작용이 이루어지며 전체 시스템이 유연하고 탄탄하게 구성될 것인가를 연구하는 과정이 설계 과정입니다.

[그림] 소프트웨어 시스템의 개발 과정

고리타분한 이야기라고 생각지 말고 필자가 이야기하는 내용들을 지금부터 습관화시키는 노력을 해보세요. 여러분이 개발한 프로그램이 훨씬 강력한 힘을 얻게 될 것입니다. 지금까지 여러분이 아무리 조그마한 프로그램을 제작했었더라도 사실은 모두 분석, 설계, 구현 과정을 거쳐서 만들어졌다고 볼 수 있습니다. 다만 그러한 과정이 여러분의 머리 속에서 모두 이루어 졌다고 할 수 있겠죠. 그러나 프로그램의 규모가 커질 경우 모든 것을 개인의 머리 속에서 처리할 수는 없습니다. 이런 경우 분석, 설계 과정의 문서화가 반드시 필요하게 되지요. 더구나 대규모 시스템 개발 또는 팀 프로젝트에서 개발 과정의 문서화는 필수 불가결한 과정이니 앞으로 큰 프로젝트 개발을 희망하는 독자라면(아마 독자 여러분들 전부가 되겠지만) 소홀히 하지 않길 바랍니다.

분석, 설계의 문서화에 대한 가이드라인으로서 독자 여러분들에게 추천하고싶은 책이 하나 있습니다. "Applying UML and Patterns -Craig Larman-"이라는 책입니다. 원서지만 영어도 쉬운 편이고 그림도 많아서 이런 내용을 처음 접하시는 분들에게는 적당한 책이 아닌가 싶습니다. 필자도 강의 교재로 처음 접했던 책인데 하나의 샘플용 소프트웨어 시스템을 개발하는 과정을 통해 개발 과정의 각 부분별 문서화 요령 등을 일관된 방식으로 익힐 수 있어서 좋았던 것으로 기억합니다. 그런 책을 구입하거나 공부하기가 부담스러우신 분들도 좀 엉성한 한이 있더라도 반드시 자신이 생각하는 내용을 지면에 옮겨놓는 습관을 들이시기 바랍니다. 아무튼 분석, 설계의 문서화는 반드시 습관들여할 사항이라 생각됩니다.

초보 프로그래머의 디버깅 노트: static VS non-static

TestClass.java:3: non-static method myMethod() cannot be referenced from a static context myMethod();

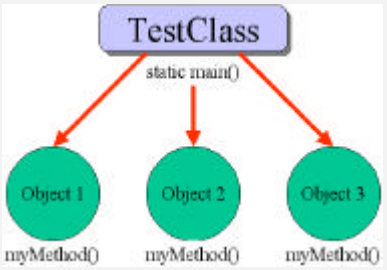
독자 여러분들 중에 컴파일할 때 위와 같은 에러 메시지를 만난 적이 있는 사람이 분명히 있을 것입니다. 자바 세계에서는 너무나도 유명한 에러이기 때문이지요. (컴파일러의 버전에 따라서는 에러 메시지가 다음과 같이 나타날 수도 있습니다: "Can't make static reference to method...") 아직껏 이런 메시지를 만나보지 못했던 독자가 있더라도 잘 보아두기 바랍니다. 자바 프로그래머라면 누구라도 수시로 만나는 에러이기 때문이죠. 더구나 초보 프로그래머들에게는 꽤나 골치 아픈 에러이기도 합니다. 그럼 이러한 에러 메시지를 만들어낸 소스코드를 살펴볼까요?

[리스트 101] static과 non-static 메쏘드를 잘 못 사용한 예

```
// TestClass.java
public class TestClass {
    public static void main(String args[]) {
        myMethod();
    }
    public void myMethod() {
        //some code
    }
}
```

한 눈에 무엇이 잘못되었는지 알아볼 수 있는 독자라면 자바 문법 지식이 꽤나 탄탄한 독자라고 할 수 있겠습니다. 즉, 이 에러를 확실하게 이해할 수 있다면 자바 문법 공부에도 상당한 도움이 된다는 얘기지요. 그럼 무엇이 잘 못되었는지 자세히 살펴볼까요?

[그림 101] static 메쏘드는 클래스의 멤버이고 non-static 메쏘드는 개개의 객체의 메쏘드다.



문제의 핵심은 static과 non-static의 차이에 있습니다. static 메쏘드 또는 static 속성(attribute)은 클래스 자체에 소속된 클래스의 멤버입니다. 이에 반해 non-static 메쏘드나 non-static 속성은 개개의 객체에 소속된 객체의 멤버입니다. 그림 [101]을 보면 static과 non-static 메쏘드의 차이를 알 수 있습니다. main()은 static이기 때문에 TestClass 자체에 속해있고 myMethod()는 TestClass가 객체화된 Object1, Object2, Object3 각각에 따로따로 존재하는 것이지요. 실제로 메쏘드를 호출할 때에는 다음과 같이 합니다.

```
TestClass.main();
Object1.myMethod();
Object2.myMethod();
Object3.myMethod();
```

즉 "메쏘드의 소속명.메쏘드 이름()" 이런 형식으로 호출하는 것이지요. (main() 메쏘드이 경우 자동으로 호출되기 때문에 프로그래머가 일부러 이렇게 하지는 않겠지만 이렇게 할 수도 있습니다.)

메쏘드란 데이터를 조작하는 작업이기 때문에 자신이 조작할 데이터가 어디에 있는지 알고 있어야 합니다. 우리의 경우 myMethod()가 object1의 데이터를 조작할 것인지 아니면 object2나 object3의 데이터를 조작할 것인지를 알고 있어야한다는 것이죠. 그래서 메쏘드 호출에는 항상 소속명이 필요한 것입니다.

해결책

```
public static void main(String args[]) {
    TestClass m = new TestClass();
    m.myMethod();
}
```

첫 번째 디버깅노트는 많이 길어졌습니다. 디버깅 이전에 자바 문법에 대한 이해가 필요했기 때문입니다. 앞으로도 문법적 이해가 필요한 부분은 자세히 설명할 생각입니다. 고기를 잡아주는 것보다는 고기가 잡히는 원리를 알려주는 것이 응용력을 기르는데 도움이 될테니까요.