Java Naming Convention

아주 오래전에 읽었봤었지만 그때는 중요성도 모르고 그저 그런가보다 하고 지나갔던 코딩컨벤션  
특히 네이밍이 코딩에서 가장 어려운 일이라는 걸 느껴본 사람이라면 다시 한번 보시길...  
<http://geosoft.no/development/javastyle.html>  
가장 중요한 말은 이 모든 규칙들은 가독성을 위해서라면 위반되어도 된다는 말.

1. 필드는 ‘\_’ 접미사를 사용한다.
2. 변수의 길이는 영역과 비례한다.
3. 객체명의 의미를 메소드명에 넣지마라. line.getLength(); // NOT: line.getLineLength
4. compute, find, initialize 사용
5. 컬렉션은 복수형으로
6. 개수를 나타내는 변수는 ’n’ 접두사를 사용한다.
7. 엔터티 번호에는 ‘i’ 접두사나 ‘No’ 접미사를 사용한다.
8. 대응하는 단어를 사용한다. get/set , add/remove , create/destroy , start/stop , insert/delete , increment/decrement , old/new , begin/end , first/last , up/down , min/max , next/previous , old/new , open/close , show/hide , suspend/resume
9. 함수명은 ‘처리 후 무엇을 리턴하는지’, 프로시저명은 ‘무엇을 처리하는지’를 나타낸다.
10. 인터페이스 디폴트 구현은 ‘Default’ 접두사를 사용한다.
11. 팩토리 메소드의 이름은 ‘new’ 접두사에 반환하는 인스턴스의 클래스명으로 한다. public Point newPoint(...)
12. 메소드 선언 규칙은 <access> static abstract synchronized <unusual> final native 순서이다.
13. do-while 사용 자제
14. 조건이 복잡한 경우 임시 boolean으로 단순화 한다.
15. 정상적인 경우를 if에 놓고 예외를 else에 둔다.
16. 조건문과 한줄에 놓지 마라. 디버깅을 위해서
17. 조건문 안에 실행문이 있어서는 안된다.
18. 특정 의미를 갖는 숫자는 항상 의미를 나타내는 상수로 바꿔서 사용한다.
19. 빈 for문은 ‘;’(세미콜론)을 새 줄에 놓는다. for(<initialization>; <condition>; <update>)  ;
20. 복잡한 메소드는 주석을 달지 말고 새로 만든다.(self-documenting)
21. javadoc주석을 제외한 주석은 multi-line 주석이라고 해도 ‘//‘를 사용한다. /\* \*/주석은 디버깅용으로 언제든 제거해서 사용 할 수 있게 한다.
22. 컬렉션은 포함하는 유형을 주석으로 뒤에 넣는다. private Vector points\_; // of Point
23. 모든 public 클래스, 또 publics 클래스내의 public, protected 메소드는 javadoc을 만든다.

Java naming은 아니지만 참고할만한 <http://msdn.microsoft.com/en-us/library/xzf533w0(v=VS.71).aspx>

**Java Naming Conventions**

|  |  |  |
| --- | --- | --- |
| Identifier Type | Rules for Naming | Examples |
| Packages | 유일 패키지명의 접두어는 항상 소문자로 된 ASCII 문자로 쓰며, 현재까지의 최상위 도메인명인 com, edu, gov, mil, net, org중에 하나이거나, ISO 표준 3166 (1981)에서 지정한 두글자로 된 영문 국가 코드 중에 하나여야 한다.  패키지명의 나머지 요소들은 조직의 내부 네이밍 규칙에 따라 바뀐다. 이러한 규칙들에 따르면 어떤 디렉토리명 요소에는 부서명, 프로젝트명, 기관명, 로그인 사용자명 등이 올 수 있다. | com.sun.eng  com.apple.quicktime.v2  edu.cmu.cs.bovik.cheese |
| Classes | 클래스명은 명사여야 하고, 각 단어의 첫글자는 대문자로 한다. 클래스명은 간단해야 하며, 그 클래스를 잘 설명할 수 있어야 한다. (만약 URL이나 HTML같은 단어처럼 축약형이 더 널리 사용되는 경우가 아니라면) 이니셜명이나 축약형은 피하고 전체 단어를 사용하라. | class Raster; class ImageSprite; |
| Interfaces | 인터페이스명은 클래스명과 같은 대소문자 규칙을 적용한다. | interface RasterDelegate; interface Storing; |
| Methods | 메소드는 동사여야 하고 첫 글자는 소문자, 다음의 각 단어의 첫글자는 대문자를 사용한다. | run(); runFast(); getBackground(); |
| Variables | 변수를 제외하고, 모든 인스턴스, 클래스, 클래스 상수는 첫 글자를 소문자로 시작하는 대소문자 혼합 형태이다. 내부 단어는 대문자로 시작해야 한다. 변수명으로 언더스코어("\_")나 달러("$") 둘다 허용은 되지만, 이들 글자들로 시작할 수는 없다.  변수명은 짧지만 의미를 가져야 한다. 변수명은 기억하기 쉬워야 한다. 즉, 임의의 관찰자가 보더라도 그 사용의도를 알 수 있게 해야 한다. 임시적인 1회성(throwaway) 변수가 아니라면 한 글자로 된 변수명은 피해야 한다. 임시변수에 대한 일반적인 이름은 i, j, k, m 가 있고, 정수를 나타내는 n, 문자를 나타내는 c, d, e가 있다. | int i;  char c;  float myWidth; |
| Constants | 클래스 상수로 선언된 변수명과 ANSI 상수명은 언더스코어("\_")로 구분된 대문자 단어를 사용한다. (디버깅을 편하게 하려면 ANSI 상수는 피하는 것이 좋다.) | static final int MIN\_WIDTH = 4;  static final int MAX\_WIDTH = 999;  static final int GET\_THE\_CPU = 1; |

**Naming Conventions**

**General Naming Conventions**

| **1. 패키지 이름은 소문자로만 하세요.** |
| --- |
| mypackage, com.company.application.ui |
| 패키지 이름은 반드시 소문자로 하세요. Sun사에서도 이러한 코딩 룰을 지키고 있습니다. |

| **2. 변수 이름은 소문자로 시작하시고, 명사로 지어주세요.** |
| --- |
| line, audioSystem |
| 변수 이름은 반드시 소문자로 시작하세요. 단어가 바뀌는 부분은 대문자로 구분해주시면 됩니다. Sun사에서도 이러한 코딩 룰을 지키고 있습니다. |

| **3. 상수(final 변수) 이름은 대문자로만 쓰세요. 단어가 바뀔 때는 언더바(\_)로 구분하시면 됩니다.** |
| --- |
| MAX\_ITERATIONS, COLOR\_RED |
| Sun사에서도 이러한 코딩 룰을 지키고 있습니다.  대부분의 경우에는 상수보다는 메서드를 통해서 제공하는 편이 좀 더 좋은 선택입니다.  int getMaxIterations() // NOT : MAX\_ITERATIONS = 25  {  return 25;  } |

| **4. 메서드 이름은 소문자로 시작하시고, 동사로 지어주세요.** |
| --- |
| getName(), computeTotalWidth() |
| Sun사에서도 이러한 코딩 룰을 지키고 있습니다. |

| **5. 이름을 지을 때 되도록 약어(약자)를 모두 대문자로 하면 가독성이 떨어집니다. 첫 문자만 대문자로 하세요.** |
| --- |
| exportHtmlSource(); *// NOT: exportHTMLSource();*  openDvdPlayer(); *// NOT: openDVDPlayer();* |
| 되도록 첫 문자만 대문자로 하는 편이 낫습니다. |

| **6. 일반적인 변수들은 타입과 변수명을 같게 하세요.** |
| --- |
| void setTopic(Topic topic) *// NOT: void setTopic(Topic value)*  *// NOT: void setTopic(Topic aTopic)*  *// NOT: void setTopic(Topic t)*  void connect(Database database) *// NOT: void connect(Database db)*  *// NOT: void connect(Database oracleDB)* |
| 일반적이지 않은 변수들은 다음과 같은 룰을 적용할 수 있습니다.  Point startingPoint, centerPoint;  Name loginName; |

| **7. 사용 범위가 넓은 변수는 이름을 길게 짓고, 사용 범위가 짧은 변수는 이름을 짧게 지으세요.** |
| --- |
|  |
| 루프문에 사용되는 변수 같이 사용 범위가 짧은 변수들은 짧은 이름을 쓰세요.  일반적으로 int형 변수는 i, j, k, m, n을 쓰고, char형 변수는 c, d를 씁니다. |

| **8. object의 이름이 암시적으로 어떤 것을 표현하고 있다면 메서드 이름에 그 부분을 쓰지 마세요.** |
| --- |
| line.getLength(); *// NOT: line.getLineLength();* |
| 의미가 중복되는 부분을 쓰지 마세요. |

**Specific Naming Conventions**

| **1. get/set 메서드는 속성(변수)에 대해서 직접적인 접근을 해야할 때 쓰세요.** |
| --- |
| employee.getName();  employee.setName(name);  matrix.getElement(2, 4);  matrix.setElement(2, 4, value); |
| Sun사에서도 이러한 코딩 룰을 지키고 있습니다. |

| **2. is 키워드는 boolean 타입의 변수나 boolean 타입을 리턴하는 메서드 앞에 붙이세요.** |
| --- |
| isSet, isVisible, isFinished, isFound, isOpen |
| Sun사에서도 이러한 코딩 룰을 지키고 있습니다.  isStatus나 isFlag는 간결하지만 좋지 않은 이름입니다.  boolean 타입의 Setter 메서드는 반드시 다음과 같이 써야합니다. (이때는 get/set이 아니라 is/set으로 매칭이 됩니다.)  void setFound(boolean isFound);  경우에 따라서는 has, can, should 키워드를 쓸 수 있습니다.  boolean hasLicense();  boolean canEvaluate();  boolean shouldAbort = false; |

| **3. 어떤 것을 계산하는 경우에는 compute 키워드를 메서드 앞에 붙일 수 있습니다.** |
| --- |
| valueSet.computeAverage();  matrix.computeInverse(); |
| 어떤 것을 계산할 때 씁니다. 계산 결과가 있을 경우 result라는 키워드를 계산 결과를 저장하는 변수에 쓸 수 있습니다. |

| **4. 어떤 것을 찾을 경우에는 find 키워드를 메서드 앞에 붙일 수 있습니다.** |
| --- |
| vertex.findNearestVertex();  matrix.findSmallestElement();  node.findShortestPath(Node destinationNode); |
| 어떤 것을 찾을 때 씁니다. |

| **5. object나 어떤 확정적인 작업의 초기화를 할 경우에는 initialize 키워드를 메서드 앞에 붙일 수 있습니다.** |
| --- |
| printer.initializeFontSet(); |
| 절대 init 이라고 쓰지 마세요. |

| **6. JFC(Swing) 변수들은 element 타입(component 타입)을 변수 뒤에 붙일 수 있습니다.** |
| --- |
| widthScale, nameTextField, leftScrollbar, mainPanel, fileToggle, minLabel, printerDialog |
| 변수 이름이 길어지더라도 타입을 뒤에 적어두면 가독성이 좋아집니다. |

| **7. collection이나 배열 타입의 변수 이름은 복수 형태로 이름을 지으세요.** |
| --- |
| Collection<Point> points;  int[] values; |
| collection 타입은 저장되는 object의 복수 형태로 지으시면 되고, 일반 배열 타입은 의미를 부여하며 복수 형태로 이름을 지으시면 됩니다. |

| **8. object의 갯수를 나타낼 경우에는 n 키워드를 변수 앞에 붙일 수 있습니다.** |
| --- |
| nPoints, nLines |
| Sun사에서는 이러한 용도로 num 키워드를 사용하고 있습니다. numberOf의 약자로써 표현하였지만 실제로는 number라는 느낌이 강하기 때문에 num이라고 쓰지 마시길 바랍니다. n 키워드 또는 numberOf 키워드를 사용해서 나타내시는 것이 좀 더 좋습니다. |

| **9. 엔티티의 번호를 나타낼 경우에는 No 키워드를 변수 뒤에 붙일 수 있습니다.** |
| --- |
| tableNo, employeeNo |
| 엔티티 번호를 나타낼 때 사용합니다. 다른 방법으로는 i 키워드를 붙이는 것입니다. iTable, iEmployee 라고 해서 iterator의 역할로써 쓰는 것입니다. |

| **10. Iterator 변수들은 i, j, k 등과 같은 이름을 사용하세요.** |
| --- |
| for (Iterator i = points.iterator(); i.hasNext(); )  {  :  }  for (int i = 0; i < nTables; i++)  {  :  } |
| 주로 i를 쓰고, j, k 등은 중첩된 반복문일때 쓰세요. |

| **11. 대응되는 단어를 사용해서 이름을 지으면 좋습니다.** |
| --- |
| get/set, add/remove, create/destroy, start/stop, insert/delete,  increment/decrement, old/new, begin/end, first/last, up/down, min/max,  next/previous, old/new, open/close, show/hide, suspend/resume, etc. |
| 이렇게 하면 가독성이 좋아집니다. |

| **12. 되도록 약자(축약형)를 쓰지 마세요.** |
| --- |
| computeAverage(); *// NOT: compAvg();*  ActionEvent event; *// NOT: ActionEvent e;*  catch (Exception exception) { *// NOT: catch (Exception e) {* |
| 되도록 이러한 형태는 피하세요.  cmd instead of command comp instead of compute cp instead of copy e instead of exception init instead of initialize pt instead of point etc.  다음과 같은 많이 쓰이는 경우는 약자를 써도 됩니다.  HypertextMarkupLanguage instead of html CentralProcessingUnit instead of cpu PriceEarningRatio instead of pe etc. |

| **13. boolean 타입 변수 이름은 부정적인 의미로 쓰지 마세요.** |
| --- |
| bool isError; *// NOT: isNoError*  bool isFound; *// NOT: isNotFound* |
| 부정적인 이름을 쓰면 가독성이 떨어집니다. 예를 들어, !isNoError 라고 표기를 하면 어떤 의미인지 파악하는 데 오래 걸립니다. |

| **14. 서로 관련이 있는 상수들은 상수 앞에 공통되는 단어를 붙여주는 것이 좋다.** |
| --- |
| final int COLOR\_RED = 1;  final int COLOR\_GREEN = 2;  final int COLOR\_BLUE = 3; |
| 다음과 같이 나타낼 수도 있다.  interface Color  {  final int RED = 1;  final int GREEN = 2;  final int BLUE = 3;  }   * 스쿨쥐의 생각 :   JDK 1.5 버전 이상이라면 enum을 사용하기를 권장합니다. 하위 호환성을 위해 enum을 쓸 수 없다면 열거타입을 나타내는 패턴을 적용하기 바랍니다. 자세한 내용은 Effective Java를 참조하시기 바랍니다. |

| **15. 예외 클래스는 클래스 뒤에 Exception 키워드를 붙이세요.** |
| --- |
| class AccessException extends Exception  {  :  } |
| Sun사에서도 이러한 코딩 룰을 지키고 있습니다. |

| **16. 디폴트 인터페이스는 인터페이스 앞에 Default 키워드를 붙이세요.** |
| --- |
| class DefaultTableCellRenderer  implements TableCellRenderer  {  :  } |
| Sun사에서도 이러한 코딩 룰을 지키고 있습니다. |

| **17. Singleton 패턴을 구현한 클래스에서 인스턴스를 리턴하는 메서드의 이름은 getInstance 라고 붙이세요.** |
| --- |
| class UnitManager  {  private final static UnitManager instance\_ = new UnitManager();  private UnitManager()  {  ...  }  public static UnitManager getInstance() *// NOT: get() or instance() or unitManager() etc.*  {  return instance\_;  }  } |
| Sun사의 JDK는 이러한 관행을 일관적으로 지원하지는 않지만 자바 커뮤니티에서는 공통적으로 사용하고 있고, 또한 패턴이 적용되었다는 것을 쉽게 알려줄 수 있습니다. |

| **18. Factory 패턴을 구현한 클래스에서 인스턴스를 리턴하는 메서드 앞에 new 키워드를 쓸 수 있습니다.** |
| --- |
| class PointFactory  {  public Point newPoint(...)  {  ... }  } |
| Factory 패턴으로 인스턴스를 제공할 경우 "new클래스명" 형태로 Factory 메서드를 제공할 수 있습니다. new Point() 생성자를 재구성하여 메서드로써 인스턴스를 만들어 준다고 생각하시면 됩니다. |

| **19. 리턴 타입이 void인 메서드(procedures)의 경우에는 "무엇을 처리하는 지"를 이름에 쓰는 것이 좋으며, object 타입이나 일반 자료형과 같이 리턴 타입이 있는 메서드(functions)의 경우에는 "처리 후 무엇을 리턴하는 지"를 이름에 쓰는 것이 좋습니다.** |
| --- |
| 예제 없음 |
| 메서드가 무엇을 의도하는 지 명확하게 알려주어 가독성을 높혀줍니다. |

**Files**

| **1. 탭(TAB)이나 페이지 구분(page break) 문자와 같은 특수한 문자들은 사용하지 마세요.** |
| --- |
| 예제 없음 |
| 혼자만 쓰면 괜찮지만 여러 사람이 공동 작업할 경우 글꼴이나 탭 설정에 따라서 정렬이 다르게 나와서 가독성을 떨어뜨릴 수 있습니다.   * 스쿨쥐의 생각 :   이클립스의 경우 영역 지정 후 단축키를 통해(기본 단축키는 ctrl + i) 자신의 설정에 맞게 정렬을 할 수 있다. |

| **2. 여러 라인으로 나눈 문장은 정렬을 잘 하세요.** |
| --- |
| totalSum = a + b + c + d + e;  method(param1, param2, param3);  setText ("Long line split" + "into two parts.");  for (int tableNo = 0; tableNo < nTables;  tableNo += tableStep) { ... } |
| 일반적으로   * 콤마(,) 뒤에서 분리합니다. * 연산자 뒤에서 분리합니다. * 이전 라인의 표현식 시작 부분에 맞춰서 정렬합니다. |

**Statements**

**Package and Import Statements**

| **1. package 문은 파일의 첫번째 라인에 적어야 하며, 모든 파일은 특정 패키지에 소속되어야 합니다.** |
| --- |
| 예제 없음 |
| package 문의 위치는 Java 언어 코딩 표준에서 권고하고 있는 것입니다. 모든 파일들을 실제 패키지(디폴트 패키지보다는 이름이 있는 패키지)에 두는 것은 Java 객체 지향 프로그래밍 기법에 도움을 줍니다. |

| **2. import 문은 반드시 package 문 뒤에 나와야 합니다. 또한 기본 패키지부터 정렬해야 하며, 관련 패키지들은 그룹핑하고 빈 라인을 삽입하여 일목요연하게 정리합니다.** |
| --- |
| import java.io.IOException;  import java.net.URL;  import java.rmi.RmiServer;  import java.rmi.server.Server;  import javax.swing.JPanel;  import javax.swing.event.ActionEvent;  import org.linux.apache.server.SoapServer; |
| import 문의 위치는 Java 언어 코딩 표준에서 권고하고 있는 것입니다. 많은 import 문들을 정렬해두면 편리하게 import 문들을 검토할 수 있고, 이를 통하여 현재 이 패키지가 어떤 용도로 설계되었는지를 쉽게 파악할 수 있습니다. 또한 import 문을 그룹핑하면 관련된 정보들을 공통의 유닛으로 관리할 수 있기 때문에 복잡도를 줄일 수 있습니다.   * 스쿨쥐의 생각 :   이클립스에서는 단축키(기본 단축키는 ctrl + shift + o)를 통해 쉽게 할 수 있습니다. |

| **3. import 문을 사용할 때는 와일드 카드 문자(\*)를 쓰지 마세요.** |
| --- |
| import java.util.List; *// NOT: import java.util.\*;*  import java.util.ArrayList;  import java.util.HashSet; |
| 와일드 카드 문자를 사용하게 되면 성능상에도 약간 영향이 있으며, 같은 이름의 클래스가 서로 다른 패키지에 있을 경우 문제가 발생할 수 있습니다. 또한 import 문만 보고도 어떠한 작업을 할 것인지 가독성을 높이기 위해서라도 와일드 카드를 쓰지 않는 것이 좋습니다.   * 스쿨쥐의 생각 :   2번과 마찬가지로 쉽게 할 수 있습니다. (동일 단축키) |

**Classes and Interfaces**

| **1. Class 와 Interface 의 선언은 다음과 같은 방식으로 조직화하여 사용합니다.**  **1. Class/Interface 문서(javadoc 주석)**  **2. class 나 interface 선언문**  **3. 클래스 변수(static으로 선언된)들을 public, protected, package (접근제한자가 없는), private 순서대로 나열합니다.**  **4. 인스턴스 변수들을 public, protected, package (접근제한자가 없는), private 순서대로 나열합니다.**  **5. 생성자**  **6. 메소드(메소드에는 특별한 순서가 없습니다)** |
| --- |
| 예제 없음 |
| 각각의 클래스/인터페이스 구성요소들의 등장 위치를 지키게되면, 현재 코드 상에 어떤 요소들이 다음에 등장할 것인지 예측할 수 있게되어 가독성이 좋아집니다. |

**Methods**

| **1. 메서드의 지시자는 다음과 같은 순서로 사용한다.**  **<access> static abstract synchronized <unusual> final native** |
| --- |
| public static double square(double a); *// NOT: static public double square(double a);* |
| <access> 지시자는 public, protected, private 중 하나이고, <unusual> 부분은 volatile 과 transient 중 하나가 지정됩니다. 여기서 가장 중요한 점은 접근(access) 지시자가 맨 처음에 나타나야 한다는 것입니다. 사용할 수 있는 다양한 지시자들이 있지만, 이는 매우 중요한 사항이기 때문에 반드시 메소드 선언문에 반영되어야 합니다. 기타 지시자들의 순서는 덜 중요하지만 일관된 관례를 따르는 것이 바람직합니다. |

**Types**

| **1. 형변환은 반드시 명시적으로 해주세요.** |
| --- |
| floatValue = (int) intValue; *// NOT: floatValue = intValue;* |
| 형변환을 할 때 데이터가 손실될 경우가 있습니다. 명시적으로 하지 않는다면 이러한 것이 의도한 것인지 아니면 실수로 빠뜨린 것인지 알 수 없습니다. |

| **2. 배열 지시자([])는 변수 이름 뒤가 아니라 타입 뒤에 붙이세요.** |
| --- |
| int[] a = new int[20]; *// NOT: int a[] = new int[20]* |
| 배열은 타입의 한 속성이지 변수의 속성이 아니기 때문입니다. 어떤 이유에서인지 Java 에서는 두 가지 모두 문법적으로 허용하고 있습니다. |

**Variables**

| **1. 변수는 선언된 지점에서 초기화하며, 가능한 사용범위를 최소화하여 선언한다.** |
| --- |
| 예제 없음 |
| 이 규칙은 어느 시점에서든 변수가 유효한 값을 가진다는 것을 보장해줍니다. 종종 선언하는 시점에 변수에 유효한 값을 초기화하는 것이 불가능한 경우가 있습니다만, 그러한 경우에도 초기화하지 않은 상태로 내버려두는 것보다는 임의의 값이라도 사용하여 초기화 해두는 것이 좋습니다.   * 스쿨쥐의 생각 :   보통 숫자 형은 0 또는 0.0, 레퍼런스 형은 null을 주는 것이 일반적이다. |

| **2. 변수는 한가지의 의미가 가져야지 여러 의미로 해석되어서는 안됩니다.** |
| --- |
| 예제 없음 |
| 일관되고 유일한 의미를 부여함으로써 가독성을 향상시키고 더불어 부작용까지 예방할 수 있습니다. |

| **3. 클래스 변수(static 변수)는 public으로 지정하지 마세요.** |
| --- |
| 예제 없음 |
| public 변수들은 Java 의 정보은닉과 캡슐화 컨셉에 위배됩니다. 대신 변수를 private 으로 선언하시고 이 변수를 접근할 수 있는 메소드를 사용하게 하십시오. 한 가지 예외적인 상황은 클래스가 데이터 구조로만 사용될 때입니다. 이 경우 클래스는 어떠한 행위(메소드)도 갖지 않고 오로지 데이터를 보관하는, 즉 C++ 에서의 struct 와 동일한 형태로 사용됩니다. 이 경우에는 클래스의 인스턴스 변수들을 public 으로 선언할 수도 있습니다. |

| **4. 배열은 반드시 타입 뒤에서 정의하세요.** |
| --- |
| double[] vertex; *// NOT: double vertex[];*  int[] count; *// NOT: int count[];*  public static void main(String[] arguments)  public double[] computeVertex() |
| 이유는 앞에서 설명한 것과 같습니다. |

| **5. 변수의 생존 기간(사용 범위)를 가능한한 짧게 하세요.** |
| --- |
| 예제 없음 |
| 변수의 생존 기간을 짧게 하면 영향을 주거나 부작용이 일어나는 것에 대해서 통제하기가 쉬워집니다. |

**Loops**

| **1. for() 구문 안에는 반복에 사용되는 문장만 넣으세요.** |
| --- |
| sum = 0; // NOT: for (i = 0, sum = 0; i < 100; i++) for (i = 0; i < 100; i++) // sum += value[i];  sum += value[i]; |
| 관리하기가 편해지고 가독성을 향상시킬 수 있습니다. 무엇이 반복문을 제어하고 무엇이 반복문 내에서 사용되는지 분명하게 하세요. |

| **2. 반복문에 사용되는 변수는 반복문 바로 앞에서 초기화 하세요.** |
| --- |
| isDone = false; // NOT : bool isDone = false; while (!isDone) { // :  : // while (!isDone) { } // :  // } |
| 설명 없음 |

| **3. do-while 문은 되도록 쓰지 마세요.** |
| --- |
| 예제 없음 |
| *do .... while* 문을 사용한 모든 문장은 *while* 문을 사용하여 동일하게 바꾸어 작성할 수 있습니다. 그렇게 하면 가독성이 더욱 좋아집니다. |

| **4. 반복문 내에서 continue나 break 문을 되도록 쓰지 마세요.** |
| --- |
| 예제 없음 |
| continue나 break는 꼭 필요할 때만 쓰세요. 너무 많이 쓰면 가독성이 떨어집니다. |

**Conditionals**

| **1. 복잡한 조건식은 피하세요. 대신 임시로 boolean 변수를 사용하세요.** |
| --- |
| bool isFinished = (elementNo < 0) || (elementNo > maxElement);  bool isRepeatedEntry = (elementNo == lastElement);  if (isFinished || isRepeatedEntry) {  :  }  *// NOT:*  *if ((elementNo < 0) || (elementNo > maxElement) || elementNo == lastElement) {*  *:*  *}* |
| 디버깅 할 때도 편해지며, 의미가 더 정확하게 전달되어 가독성이 좋아집니다. |

| **2. if문에서 일반적인 상황은 if 부분에 위치시키고, 예외적인 상황은 else 부분에 위치시키세요.** |
| --- |
| boolean isOk = readFile(fileName);  if (isOk) {  :  }  else {  :  } |
| 일반적인 처리 흐름과 예외상황 처리 흐름을 불명확하지 않게 하십시오. 이 지침은 가독성과 성능 모두에 영향을 미치는 중요한 사항입니다. |

| **3. 조건식에 실행문을 적지 마세요.** |
| --- |
| InputStream stream = File.open(fileName, "w");  if (stream != null) {  :  }  *// NOT:*  *if (File.open(fileName, "w") != null)) {*  *:*  *}* |
| 조건식에 실행문을 사용하는 것은 간편하지만 읽기가 매우 어려워집니다. 이 지침은 Java 에 발을 처음으로 들여놓는 개발자들에게 특히 강조하고 싶은 지침입니다. |

**Miscellaneous**

| **1. 코드 상에 매직 넘버를 쓰지 마세요. 0과 1 이외의 숫자는 상수로 정의해서 쓰는 것이 좋습니다.** |
| --- |
| private static final int TEAM\_SIZE = 11;  :  Player[] players = new Player[TEAM\_SIZE]; *// NOT: Player[] players = new Player[11];* |
| 좀 더 의미를 명확하게 하게 할 수 있으며, 재사용성도 좋아집니다. |

| **2. 실수값 상수는 항상 소수점과 최소 소수점 이하 한자리 숫자를 사용하여 지정해야 합니다.** |
| --- |
| double total = 0.0; *// NOT: double total = 0;*  double speed = 3.0e8; *// NOT: double speed = 3e8;*  double sum;  :  sum = (a + b) \* 10.0; |
| 특별한 경우에 정수와 실수가 동일한 값을 갖는다 하더라도 기본적으로 특성이 다르기 때문에 구별하는 것이 좋습니다.  또한 위 마지막 라인의 예제에서 살펴볼 수 있듯이, 값을 할당받은 변수(sum)는 다른 변수의 타입을 알 수 없다고 하더라도 명백하게 실수값을 계산해 낸다는 것을 보장할 수 있습니다. |

| **3. 실수값 상수는 항상 정수부에 숫자를 사용하여 지정해야 합니다.** |
| --- |
| double total = 0.5; *// NOT: double total = .5;* |
| Java 에서 사용하는 숫자와 표현식은 수학식 표기법에서 차용하였으므로, 프로그래머는 가능한 수학적 표기법을 준수하는 것이 바람직합니다. 실제로 .5 (간편한 표기) 보다 0.5 (수학적 표기) 가 더 읽기에 쉽습니다. 이 수식에서 정수 5 가 함께 사용되는 경우에는, .5 와 정수 5 를 혼동하여 잘못된 연산을 초래할 가능성이 높아지게 됩니다. |

| **4. static 변수나 메서드는 인스턴스 변수를 통해서 호출하지 말고, 클래스 이름을 통해서 접근해야 합니다.** |
| --- |
| Thread.sleep(1000); *// NOT: thread.sleep(1000);* |
| 특정 인스턴스에 대해서 독립적이라는 것을 강조하기 위해서 사용합니다. 즉, static을 사용하고 있다는 것을 알려주어 가독성을 높여줍니다. |

**Layout and Comments**

**Layout**[**#**](http://scroogy.springnote.com/pages/1023608#toc_14)

| **1. 블락 레이아웃은 다음 예제를 따르세요.** |
| --- |
| while (!done) {  doSomething();  done = moreToDo();  }  while (!done)  {  doSomething();  done = moreToDo();  }  *// NOT:*  *while (!done)*  *{*  *doSomething();*  *done = moreToDo();*  *}* |
| 첫번째나 두번째 방법 중 아무 것이나 써도 됩니다. 다만 마지막 방식으로는 쓰지 마세요. |

| **2. 클래스와 인터페이스 선언은 다음 예제를 따르세요.** |
| --- |
| class Rectangle extends Shape  implements Cloneable, Serializable  {  ...  } |
| extends 키워드는 클래스나 인터페이스 뒤에 붙이시고, implements 키워드는 다음 라인에 적으세요. |

| **3. 메서드 선언은 다음 예제를 따르세요.** |
| --- |
| public void someMethod()  throws SomeException  {  ...  } |
| throws 키워드는 다음 라인에 적으세요. |

| **4. if문이나 반복문의 경우 한 줄만 있더라도 블락을 꼭 써주세요.** |
| --- |
| if(condition)  {  //statement;  }  *// NOT:*  if(condition)  //statement; |
| 한 줄이라도 블락을 쓰는 쪽이 가독성이 좋습니다. 또한 짧은 구문이라면 한줄로 적으셔도 괜찮습니다. |

**White Space**

| **1. 논리 단위로 빈 라인을 넣어주세요.** |
| --- |
| *// Create a new identity matrix*  Matrix4x4 matrix = new Matrix4x4();  *// Precompute angles for efficiency*  double cosAngle = Math.cos(angle);  double sinAngle = Math.sin(angle);  *// Specify matrix as a rotation transformation*  matrix.setElement(1, 1, cosAngle);  matrix.setElement(1, 2, sinAngle);  matrix.setElement(2, 1, -sinAngle);  matrix.setElement(2, 2, cosAngle);  *// Apply rotation*  transformation.multiply(matrix); |
| 마치 국어에서 문단을 나누듯이 논리 단위가 바뀔 때마다 빈 라인을 넣어주세요. |

| **2. 변수와 데이터 타입을 구분해서 맞추세요.** |
| --- |
| TextFile file;  int nPoints;  double x, y; |
| 이런 식으로 정렬하면 가독성이 좋아집니다. |

| **3. 구문(statement)는 가독성이 좋게 정렬하세요.** |
| --- |
| if (a == lowValue) compueSomething();  else if (a == mediumValue) computeSomethingElse();  else if (a == highValue) computeSomethingElseYet();  value = (potential \* oilDensity) / constant1 +  (depth \* waterDensity) / constant2 +  (zCoordinateValue \* gasDensity) / constant3;  minPosition = computeDistance(min, x, y, z);  averagePosition = computeDistance(average, x, y, z);  switch (phase) {  case PHASE\_OIL : text = "Oil"; break;  case PHASE\_WATER : text = "Water"; break;  case PHASE\_GAS : text = "Gas"; break;  } |
| 가독성이 좋아진다면 어느 정도 코딩 룰을 바꾸어 적용해도 괜찮습니다. |

**Comments**[**#**](http://scroogy.springnote.com/pages/1023608#toc_16)

| **1. 프로젝트 진행시 다음 주석 스타일을 따릅니다.** |
| --- |
| 1. package example;  import java.util.ArrayList; import java.util.Collection;  /\*\* \* JavaDoc 테스트용 클래스입니다. \* \* @author 최영목 \* @since 2008.04.03 \* @version 1.0 \* <pre> \* 2008.04.03 : 최초 작성 \* 2008.04.04 : 컬렉션 추가 \* </pre> \*/ public class DocExample {  private Collection<String> userNames;  public DocExample(String name) { userNames = new ArrayList<String>(); }  /\*\* \* 유저 이름을 삭제하는 메서드 \* \* @param name 삭제할 유저 이름 \* @return 유저가 존재해서 삭제가 되면 true를 리턴하고, 유저가 존재하지 않으면 false를 리턴한다. \* @throws Exception 그냥 테스트용으로 했음. \*/ public boolean removeUserName(String name) throws Exception{ return userNames.remove(name); }  /\*\* \* 어노테이션과 함께 쓸 때를 위한 예제 메서드 \* \* @return 그냥 텍스트가 출력됨 \*/ @Override public String toString() { return "테스트용입니다."; }  } |
| 이클립스의 기능에 의존하여 코드 가독성을 높이기 위해 평소에는 주석을 숨겨두고 필요할 때만 꺼내씁니다. 제너릭 타입은 반드시 명시를 해야합니다. |

| **2. JavaDoc 생성시 package 설명을 위해 각 패키지에 package-info.java 파일을 만듭니다.** |
| --- |
| 1. /\*\* \* 설명입니다. \*/  package example; |
| 반드시 /\*\* \*/ 주석이 먼저 와야하며 뒤에 패키지 선언이 와야 합니다. 또한 이 파일은 1.5에서부터 사용가능하며, 이전 버전은 doc 폴더 내의 각 패키지에 package.html 또는 package-info.html 파일을 만들어서 넣어야 합니다 |

출처: <http://aesthetica7.tistory.com/entry/Java-Naming-Convention-펌-시간-날-때-읽어-보면-매우-좋을-듯-하다> [WWW.오왕근.ORG]