Javadoc 도구에 대한 문서 주석을 작성하는 방법

Javadoc 홈 페이지

이 문서는 Java Software, Oracle에서 작성된 Java 프로그램에 대한 문서 주석에서 사용하는 스타일 가이드, 태그 및 이미지 규칙을 설명합니다. 다른 곳에서 다루는 관련 자료는 다시 해시하지 않습니다.

- Javadoc 태그에 대한 참조 자료는 Javadoc 참조 페이지를 참조 하십시오 .
- 문서 주석 의 필수 의미 론적 내용 은 Java API 사양 작성을위한 요구 사항을 참조하십시오 .

내용

소개

원칙

Java Software에는 문서 주석을 타사 개발자와 다르게 만들 수있는 몇 가지 지침이 있습니다. 문서 주석은 공식 *Java* 플랫폼 *API* 사양을 정의합니다 . 이를 위해 우리의 대상은 개발자 외에 Java 호환성 테스트를 작성하거나 Java 플랫폼을 준수 또는 재 구현하는 사람들입니다. 우리는 일반적인 프로그래밍 용어를 정의하고 개념적 개요를 작성하고 개발자를위한 예제를 포함하는 대신 경계 조건, 인수 범위 및 코너 케이스를 지정하는 데 시간과 노력을 쏟습니다.

따라서 일반적으로 문서 주석을 작성하는 방법에는 API 사양 또는 프로그래밍 가이드 문서의 두 가지 방법이 있습니다. 이 두 대상은 다음 섹션에서 설명합니다. 풍부한 리소스를 가진 직원은 두 가지를 동일한 문서 (적절하게 "청크")로 혼합 할 수 있습니다. 그러나 우선 순위에 따라 문서 주석에 API 사양을 작성하는 데 중점을 둡니다. 이것이 개발자 가 프로그래밍 가이드를 위해 Java SE 기술 문서 및 Java Tutorials 와 같은 다른 문서를 자주 찾아야하는 이유입니다 .

API 사양 작성

이상적으로는 Java API 사양은 "한 번 작성하면 어디서나 실행"을위한 Java 플랫폼의 클린 룸 구현을 수행하는 데 필요한 모든 어설 션으로 구성되어 모든 Java 애플릿 또는 응용 프로그램이 모든 구현에서 동일하게 실행됩니다. 여기에는 문서 주석의 주장과 아키텍처 및 기능 사양 (일반적으로 FrameMaker로 작성) 또는 기타 문서의 주장이 포함될 수 있습니다. 이 정의는 높은 목표이며 API를 얼마나 완벽하게 지정할 수 있는지에 대한 실질적인 제한이 있습니다. 다음은 우리가 따르려고하는 원칙입니다.

- **Java Platform API 사양은 소스 코드의 문서 주석과 해당 주석에서 도달 할 수있는 사양으로 표시된 문서에 의해 정의됩니다.**

  사양이 문서 주석에 완전히 포함될 필요는 없습니다. 특히 긴 사양은 때때로 별도의 파일로 가장 잘 형식화되고 문서 주석에서 링크됩니다.

- **Java Platform API 사양은 호출자와 구현 간의 계약입니다.**

  사양은 호출자가 의존 할 수있는 각 메서드 동작의 모든 측면을 설명합니다. 메서드가 기본인지 동기화되었는지 여부와 같은 구현 세부 정보는 설명하지 않습니다. 사양은 주어진 객체가 제공하는 스레드 안전성 보장을 (텍스트로) 설명해야합니다. 반대로 명시적인 표시가없는 경우 모든 객체는 "스레드 안전"으로 간주됩니다 (즉, 여러 스레드가 동시에 액세스 할 수 있음). 현재 사양이 항상이 이상에 부합하는 것은 아닙니다.

- **달리 명시되지 않는 한 Java API 사양 어설 션은 구현 독립적이어야합니다. 예외는 별도로 설정되어야하며 그와 같이 눈에 잘 띄게 표시되어야합니다.**

  구현 차이점을 눈에 띄게 문서화하는 방법에 대한 지침이 있습니다.

- **Java API 사양에는 Software Quality Assurance가 완전한 JCK (Java Compatibility Kit) 테스트를 작성할 수 있도록 충분한 어설 션이 포함되어야합니다.**

  이는 문서 주석이 SQA에 의한 적합성 테스트의 요구를 충족해야 함을 의미합니다. 주석은 버그 또는 현재 사양을 벗어난 구현이 작동하는 방식을 문서화해서는 안됩니다.

프로그래밍 가이드 문서 작성

프로그래밍 가이드에서 API 사양을 구분하는 것은 예, 일반적인 프로그래밍 용어의 정의, 특정 개념 개요 (예 : 은유), 구현 버그 및 해결 방법에 대한 설명입니다. 이것이 개발자의 이해에 기여하고 개발자가 신뢰할 수있는 응용 프로그램을 더 빨리 작성하는 데 도움이된다는 데에는 이견이 없습니다. 그러나 여기에는 API "어설 션"이 포함되어 있지 않으므로 API 사양에 필요하지 않습니다. 이 정보의 일부 또는 전부를 문서 주석에 포함 할 수 있으며 사용자 정의 태그를 포함 할 수 있습니다., 사용자 정의 doclet에 의해 처리됨). Java Software에서 우리는 의도적으로 문서 주석에이 수준의 문서를 포함하지 않고 대신이 정보에 대한 링크 (Java Tutorial 링크 및 변경 사항 목록)를 포함하거나 API 사양과 동일한 문서 다운로드 번들에이 정보를 포함합니다. -JDK 문서 번들에는 API 사양과 데모, 예제 및 프로그래밍 가이드가 포함되어 있습니다.

버그 및 해결 방법을 문서화하는 방법에 대해 자세히 살펴 보는 것이 유용합니다. 코드가 어떻게 사이에 차이가 때때로이 있어야 일을하고 어떻게 실제로 작동은. 이는 API 사양 버그와 코드 버그의 두 가지 다른 형태를 취할 수 있습니다. 문서 주석에 문서화할지 여부를 미리 결정하는 것이 유용합니다. Java Software에서 우리는 예외가 있지만이 두 가지를 문서 주석 외부에 문서화하기로 결정했습니다.

API 사양 버그 는 구문 또는 의미 체계에 영향을주는 메서드 선언 또는 문서 주석에있는 버그입니다. 이러한 사양 버그의 예 `null`는 전달 될 때 NullPointerException을 발생 시키도록 지정된 메서드 이지만 `null`실제로는 허용되어야하는 유용한 매개 변수입니다. API 사양을 수정하기로 결정한 경우 API 사양 자체 나 사양 변경 목록 또는 둘 다에 명시하는 것이 유용합니다. 이와 같은 API 차이점을 해결 방법과 함께 문서 주석에 문서화하면 개발자에게 변경 사항이 가장 많이 나타날 가능성이 있음을 알립니다. 이 수정 사항이있는 API 사양은 구현과 무관하게 유지됩니다.

코드 버그 는 API 사양이 아닌 구현의 버그입니다. 코드 버그와 그 해결 방법은 종종 버그 보고서에서 별도로 배포됩니다. 그러나 Javadoc 도구를 사용하여 특정 구현에 대한 문서를 생성하는 경우 문서 주석에이 정보를 포함하고 메모 또는 사용자 정의 태그 (예 :)로 적절하게 구분하는 것이 매우 유용합니다 `@bug`.

문서 주석을 소유하고 편집하는 사람

Java 플랫폼 API 사양에 대한 문서 주석은 프로그래머 소유입니다. 그러나 이들은 프로그래머와 작가 모두에 의해 편집됩니다. 작가와 프로그래머가 서로의 능력을 존중하고 둘 다 가능한 최고의 문서 주석에 기여하는 것이 기본 전제입니다. 지식, 시간, 리소스, 관심, API 복잡성 및 구현 자체 상태를 기반으로 문서의 어떤 부분을 작성하는지 결정하는 것은 종종 협상의 문제입니다. 그러나 최종 의견은 담당 엔지니어의 승인을 받아야합니다.

이상적으로는 API를 설계하는 사람은 선언과 문서 주석 만 사용하여 API 사양을 스켈레톤 소스 파일에 작성하고 작성된 API 계약을 충족하기 위해서만 구현을 채 웁니다. API 작성자의 목적은이 작업의 일부에서 디자이너를 덜어주는 것입니다. 이 경우 API 디자이너는 희소 언어를 사용하여 초기 문서 주석을 작성한 다음 작성자가 주석을 검토하고 내용을 수정하고 태그를 추가합니다.

문서 주석이 단순히 개발자를위한 가이드가 아니라 재구 현자를위한 API 사양 인 경우, API를 설계하고 구현 한 프로그래머 나 주제 전문가이거나 그에 해당하는 API 작성자가 작성해야합니다. . 구현이 사양에 작성되었지만 문서 주석이 완료되지 않은 경우 작성자는 소스 코드를 검사하거나 API를 테스트하는 프로그램을 작성하여 문서 주석을 완료 할 수 있습니다. 작성기는 throw 된 예외, 매개 변수 경계 조건 및 널 인수 허용을 검사하거나 테스트 할 수 있습니다. 그러나 구현이 그렇지 않은 경우 훨씬 더 어려운 상황이 발생합니다.사양에 기록되었습니다. 그런 다음 작성자는 디자이너의 의도를 알고 있거나 (디자인 회의 또는 별도로 작성된 디자인 사양을 통해) 디자이너에게 질문을 할 준비가 된 경우에만 API 사양을 작성할 수 있습니다. 따라서 작성자가 구현자가없는 인터페이스 및 추상 클래스에 대한 문서를 작성하는 것이 더 어려울 수 있습니다.

이를 염두에두고이 지침은 완성 된 문서 주석을 설명하기위한 것입니다. 지나치게 부담스러워 보이거나 창의적인 대안을 찾을 수있는 경우 강제로 따라야하는 요구 사항이 아니라 제안 으로 의도 된 것입니다. Java (약 60 개의 패키지 포함)와 같은 복잡한 시스템이 개발 될 때 종종 특정 패키지 세트에 기여하는 엔지니어 그룹이 `javax.swing`다른 그룹과 다른 지침을 개발할 수 있습니다. 이는 해당 패키지의 요구 사항이 다르거 나 리소스 제약 때문일 수 있습니다.

술어

API 문서 (API 문서) 또는 **API 사양** (API 사양)

주로 Java로 작성하는 프로그래머를위한 API의 온라인 또는 하드 카피 설명입니다. Javadoc 도구를 사용하여 생성하거나 다른 방법으로 생성 할 수 있습니다. API 사양은 위에 설명 된 특정 종류의 API 문서 입니다. API 사양의 예는 온라인 Java 플랫폼, Standard Edition 7 API 사양 입니다.

문서 주석 (문서 주석)

구분 기호로 구분 된 Java 소스 코드의 특수 주석입니다 `/** ... */`. 이러한 주석은 Javadoc 도구에서 처리되어 API 문서를 생성합니다.

javadoc

문서 주석에서 API 문서를 생성하는 JDK 도구입니다.

소스 파일

Javadoc 도구는 네 가지 유형의 "소스"파일에서 생성되는 출력을 생성 할 수 있습니다.

- Java 클래스 (.java) 용 소스 코드 파일-여기에는 클래스, 인터페이스, 필드, 생성자 및 메소드 주석이 포함됩니다.
- 패키지 주석 파일-여기에는 패키지 주석이 포함됩니다.
- 개요 주석 파일-패키지 세트에 대한 주석을 포함합니다.

- 기타 처리되지 않은 파일-여기에는 이미지, 샘플 소스 코드, 클래스 파일, 애플릿, HTML 파일 및 이전 파일에서 참조하려는 기타 파일이 포함됩니다.

자세한 내용은 소스 파일을 참조하십시오 .

문서 주석 작성

문서 주석 형식

문서 주석은 HTML로 작성되며 클래스, 필드, 생성자 또는 메서드 선언 앞에 와야합니다. 두 부분으로 구성됩니다. 설명 뒤에 블록 태그가 있습니다. 이 예에서, 블록 태그는 `@param`, `@return`하고 `@see`.

```
/**
* Returns an Image object that can then be painted on the screen.
* The url argument must specify an absolute <a href="#{@link}">{@link URL}</a>. The na
* argument is a specifier that is relative to the url argument.
* <p>
* This method always returns immediately, whether or not the
* image exists. When this applet attempts to draw the image on
* the screen, the data will be loaded. The graphics primitives
* that draw the image will incrementally paint on the screen.
*
* @param  url  an absolute URL giving the base location of the image
* @param  name the location of the image, relative to the url argument
* @return      the image at the specified URL
* @see         Image
*/
public Image getImage(URL url, String name) {
try {
return getImage(new URL(url, name));
} catch (MalformedURLException e) {
return null;
}
}
```

메모:

- 결과 HTML 실행 자바 독에서는 다음과 같습니다
- 위의 각 줄은 주석 아래의 코드에 맞게 들여 쓰기됩니다.
- 첫 번째 줄에는 주석 시작 구분 기호 (`/**`)가 포함됩니다.
- Javadoc 1.4부터 선행 별표는 선택 사항 입니다.
- Javadoc이 자동으로 메소드 요약 테이블 (및 색인)에 배치하므로 첫 번째 문장을 메소드의 간단한 요약으로 작성하십시오.
- `{@link URL}`URL 클래스에 대한 문서를 가리키는 HTML 하이퍼 링크로 변환되는 인라인 태그 를 확인하십시오. 이 인라인 태그는 블록 태그 다음에 오는 텍스트와 같이 주석을 쓸 수있는 모든 곳에서 사용할 수 있습니다.
- 문서 주석에 두 개 이상의 단락이있는 경우 `<p>`그림과 같이 단락 태그로 단락을 구분합니다 .
- 그림과 같이 설명과 태그 목록 사이에 빈 주석 줄을 삽입합니다.
- "@"문자로 시작하는 첫 번째 줄은 설명을 끝냅니다. 문서 주석 당 하나의 설명 블록 만 있습니다. 블록 태그 다음에 설명을 계속할 수 없습니다.
- 마지막 줄에는 주석 끝 구분 기호 (`*/`)가 있습니다. 주석 시작 구분 기호와 달리 끝 주석에는 별표가 하나만 포함됩니다.

더 많은 예는 간단한 예를 참조하십시오 .

따라서 줄 바꿈이되지 않고 문서 주석 줄을 80 자로 제한합니다.

다음은 Javadoc 도구를 실행 한 후 이전 예제의 모습입니다.

getImage

```
public Image getImage(URL url,
              String name)
```

`Image`화면에 그릴 수 있는 객체를 반환합니다 . `url`인수는 절대 URL을 지정해야합니다. `name`인수는에 상대적인 지정 자입니다 `url`인수입니다.

이 메서드는 이미지가 있는지 여부에 관계없이 항상 즉시 반환합니다. 이 애플릿이 화면에 이미지를 그리려고하면 데이 터가로드됩니다. 이미지를 그리는 그래픽 프리미티브는 화면에 점진적으로 페인트됩니다.

매개 변수 :

`url` -이미지의 기본 위치를 제공하는 절대 URL.

`name`- `url`인수에 상대적인 이미지의 위치 .

보고:

지정된 URL의 이미지

또한보십시오:

`Image`

또한 이 문서의 끝에있는 곱슬 따옴표 문제 해결 (Microsoft Word) 을 참조하십시오 .

문서 주석 검사 도구

Oracle에서는 문서 주석을 확인하기위한 도구 인 Oracle Doc Check Doclet 또는 DocCheck를 개발했습니다. 소스 코드에서 실행하고 주석에 어떤 스타일과 태그 오류가 있는지 설명하는 보고서를 생성하고 변경을 권장합니다. 우리는 그 규칙이이 문서의 규칙을 따르도록 노력했습니다.

DocCheck는 Javadoc doclet 또는 "플러그인"이므로 Javadoc 도구가 설치되어 있어야합니다 (Java 2 Standard Edition SDK의 일부로).

설명

첫 문장

각 문서 주석의 첫 번째 문장은 API 항목에 대한 간결하지만 완전한 설명을 포함하는 요약 문장이어야합니다. 이것은 각 멤버, 클래스, 인터페이스 또는 패키지 설명의 첫 번째 문장을 의미합니다. Javadoc 도구는이 첫 번째 문장을 적절한 멤버, 클래스 / 인터페이스 또는 패키지 요약에 복사합니다. 따라서 독자적으로 설 수있는 선명하고 유익한 초기 문장을 작성하는 것이 중요합니다.

이 문장은 공백, 탭 또는 줄 종결자가 뒤 따르는 첫 번째 마침표 또는 첫 번째 태그 (아래에 정의 됨)에서 끝납니다. 예를 들어,이 첫 번째 문장은 "Prof."로 끝납니다.

```
/**
* This is a simulation of Prof. Knuth's MIX computer.
*/
```

그러나 마침표 바로 뒤에 "&"또는 "<"와 같은 HTML 메타 문자를 입력하여이 문제를 해결할 수 있습니다.

```
/**
* This is a simulation of Prof. Knuth's MIX computer.
*/
```

또는

```
/**
* This is a simulation of Prof.<!-- --> Knuth's MIX computer.
*/
```

특히 오버로드 된 메서드를 구분하는 요약문을 작성합니다. 예를 들면 :

```
/**
* Class constructor.
*/
foo() {
...

/**
* Class constructor specifying number of objects to create.
*/
foo(int n) {
...
```

구현 독립성

구현에 독립적 인 설명을 작성하되 필요한 경우 이러한 종속성을 지정하십시오. 이는 엔지니어가 "한 번 작성하면 어디서나 실행"되도록 코드를 작성하는 데 도움이됩니다.

- 가능한 한 구현 독립적 인 API 사양으로 문서 주석을 작성하십시오.
- 필요한 사항과 플랫폼 / 구현에 따라 달라지는 사항을 명확히 정의하십시오.
- 이상적으로는 구현자를 준수하기에 충분하도록 완료하십시오. 현실적으로 소스 코드를 읽는 사람이 상당한 적합성 테스트를 작성할 수 있도록 충분한 설명을 포함하십시오. 기본적으로 경계 조건, 매개 변수 범위 및 코너 케이스를 포함하여 사양이 완료되어야합니다.

- 적절한 경우 사양이 지정되지 않은 상태로 두거나 구현에 따라 달라질 수있는 사항을 언급합니다.
- 구현 별 동작을 문서화해야하는 경우 구현 별 동작임을 명확히하는 도입 문구와 함께 별도의 단락에 문서화하십시오. 플랫폼에 따라 구현이 다른 경우 단락 시작 부분에 "On <platform>"을 지정하십시오. 플랫폼의 구현에 따라 달라질 수 있는 다른 경우에는 "구현 별 :"이라는 도입 문구를 사용할 수 있습니다. 다음은에 대한 사양의 구현 종속 부분의 예입니다 `java.lang.Runtime`.

Windows 시스템에서 `loadLibrary`메서드 의 경로 검색 동작은 Windows API의 `LoadLibrary`절차 와 동일 합니다.

문장 시작 부분에 "On Windows"를 사용하면 이것이 구현 참고 사항임을 분명히 알 수 있습니다.

메서드 주석의 자동 재사용

Javadoc 도구가 다른 메소드를 대체하거나 구현하는 메소드에 대한 주석을 복제 (상속)하는 방법을 인식하면 문서 주석을 다시 입력하지 않아도됩니다. 이것은 세 가지 경우에 발생합니다.

- 클래스의 메서드가 수퍼 클래스의 메서드를 재정의하는 경우
- 인터페이스의 메서드가 수퍼 인터페이스의 메서드를 재정의하는 경우
- 클래스의 메소드가 인터페이스의 메소드를 구현하는 경우

처음 두 경우에서 메소드 `m()`가 다른 메소드를 대체 하는 경우 Javadoc 도구는 대체 `m()`하는 메소드에 대한 링크와 함께 에 대한 문서에 "재정의"라는 부제목을 생성합니다 .

세 번째 경우, `m()`주어진 클래스의 메소드가 인터페이스에서 메소드를 구현하는 경우 Javadoc 도구는 구현중인 메소드에 대한 `m()`링크와 함께 에 대한 문서에서 "지정한"부제목을 생성합니다 .

이 세 가지 경우 모두 메소드 `m()`에 문서 주석이나 태그가 없으면 Javadoc 도구는 대체하거나 구현하는 메소드의 텍스트를 .NET 용 생성 된 문서에 복사합니다 `m()`. 따라서 재정의되거나 구현 된 메서드의 문서가 충분하다면 .NET Framework 에 대한 문서를 추가 할 필요가 없습니다 `m()`. 에 문서 주석 또는 태그를 추가하는 경우 `m()`"재정의"또는 "지정한 사람"부제목 및 링크는 계속 표시되지만 텍스트는 복사되지 않습니다.

스타일 가이드

다음은 문서 주석에 설명을 작성하는 데 유용한 팁과 규칙입니다.

- **키워드와 이름에 <code> 스타일을 사용하십시오.**
  설명에서 언급 될 때 키워드와 이름은 <code> ... </ code>로 오프셋됩니다. 여기에는 다음이 포함됩니다.
  - 자바 키워드
  - 패키지 이름
  - 클래스 이름
  - 메서드 이름
  - 인터페이스 이름
  - 필드 이름
  - 인수 이름
  - 코드 예
- **인라인 링크를 경제적으로 사용**
  태그를 사용하여 API 이름 (바로 위에 나열 됨)에 대한 링크를 추가하는 것이 좋습니다 . 문서 주석 에 모든 API 이름에 대한 링크를 추가 할 필요는 없습니다 . 링크는 자신의주의를 끌기 때문에 (HTML의 색상과 밑줄, 소스 코드 문서 주석의 길이) 주석을 많이 사용하면 읽기가 더 어려워 질 수 있습니다. 따라서 다음과 같은 경우 API 이름에 링크를 추가하는 것이 좋습니다. `{@link}`
  - 사용자는 (귀하의 판단에 따라) 더 많은 정보를 얻기 위해 실제로 클릭하기를 원할 수 있습니다.

- 문서 주석에서 각 API 이름이 처음 나타나는 경우에만 (링크를 반복하지 마십시오)

우리의 청중은 (초보가 아닌) 고급 프로그래머이므로 일반적으로 java.lang 패키지 (예 : String)의 API 또는 잘 알려진 다른 API에 연결할 필요가 없습니다.

- **일반 형식의 메서드 및 생성자에 대한 괄호 생략**
  여러 형식이있는 메서드 또는 생성자를 참조 할 때 특정 형식을 참조하려면 괄호와 인수 유형을 사용하십시오. 예를 들어 ArrayList에는 add (Object) 및 add (int, Object)의 두 가지 추가 메서드가 있습니다.

  `add(int, Object)` 있어서의 ArrayList의 지정된 위치에 항목을 추가한다.

  그러나 두 가지 형식의 메서드를 모두 참조하는 경우 괄호를 모두 생략하십시오. 빈 괄호를 포함하는 것은 특정 형식의 메서드를 암시하기 때문에 잘못된 것입니다. 여기서 의도는 일반적인 방법을 특정 형식과 구별하는 것입니다. 필드가 아닌 방법으로 구분하려면 "방법"이라는 단어를 포함합니다.

  이 `add`방법을 사용하면 항목을 삽입 할 수 있습니다. (선호)

  이 `add()` 방법을 사용하면 항목을 삽입 할 수 있습니다. (add 메소드의 "모든 형태"를 의미하는 경우 사용하지 마십시오)

- **간결성을 위해 완전한 문장 대신 구문을 사용하는 것이 좋습니다.**
  이것은 특히 초기 요약과 @param 태그 설명에서 유지됩니다.

- **2 인칭 (규범 적)이 아닌 3 인칭 (설명 적)을 사용하십시오.**

설명은 2 인칭 명령이 아닌 3 인칭 선언입니다.

레이블을 가져옵니다. (선호)

라벨을 받으세요. (기피)

- **메서드 설명은 동사 구로 시작됩니다.**

메서드는 작업을 구현하므로 일반적으로 동사 구로 시작합니다.

이 버튼의 레이블을 가져옵니다. (선호)

이 메서드는이 버튼의 레이블을 가져옵니다.

- **클래스 / 인터페이스 / 필드 설명은 주제를 생략하고 단순히 객체를 나타낼 수 있습니다.**

이러한 API는 종종 작업이나 동작보다는 사물을 설명합니다.

버튼 레이블. (선호)

이 필드는 버튼 레이블입니다. (기피)

- **현재 클래스에서 생성 된 객체를 참조 할 때 "the"대신 "this"를 사용하십시오.**

예를 들어 `getToolkit`메서드에 대한 설명은 다음과 같아야합니다.

이 컴포넌트의 툴킷을 가져옵니다. (선호)

구성 요소의 툴킷을 가져옵니다. (기피)

- **API 이름 외에 설명을 추가합니다.**

최고의 API 이름은 "자체 문서화"입니다. 즉, 기본적으로 API의 기능을 알려줍니다. 문서 주석이 단순히 API 이름을 문장 형식으로 반복하는 경우 더 많은 정보를 제공하지 않습니다. 예를 들어, 메소드 설명이 메소드 이름에 나타나는 단어 만 사용하는 경우 추론 할 수있는 항목에 아무것도 추가하지 않습니다. 이상적인 주석은 이러한 단어를 넘어서며 API 이름에서 즉시 명확하지 않은 약간의 정보로 항상 보상해야합니다.

피하십시오 -아래 설명은 메소드 이름을 읽음으로써 아는 것 외에는 아무것도 말하지 않습니다. "set", "tool", "tip"및 "text"라는 단어가 문장에서 반복됩니다.

```
/**
* Sets the tool tip text.
*
* @param text  the text of the tool tip
*/
public void setToolTipText(String text) {
```

선호 됨 -이 설명은 등록 및 커서에 대한 응답으로 표시되는 더 큰 컨텍스트에서 도구 설명이 무엇인지 더 완벽하게 정의합니다.

```
/**
* Registers the text to display in a tool tip.   The text
* displays when the cursor lingers over the component.
*
* @param text  the string to display.  If the text is null,
*              the tool tip is turned off for this component.
*/
public void setToolTipText(String text) {
```

- **"필드"라는 용어를 사용할 때는 명확해야합니다.**
  "필드"라는 단어에는 두 가지 의미가 있습니다.
- "클래스 변수"의 또 다른 용어 인 정적 필드
- TextField 클래스에서와 같이 텍스트 필드. 이러한 종류의 필드는 날짜, 숫자 또는 텍스트를 보유하는 것으로 제한 될 수 있습니다. 대체 이름은 "날짜 필드"또는 "숫자 필드"일 수 있습니다.
- **라틴어를 피하십시오**

"aka"대신 "일명"을 사용하고, "ie"대신 "that is"또는 "to be specific"을 사용하고, "eg"대신 "for example"을 사용하고, "in other words"또는 "즉, "viz"대신 "

태그 규칙

이 섹션에서는 다음을 다룹니다.

- 태그 순서
- 여러 태그 주문
- 필수 태그
- 태그 설명 : @author @version @param @return @deprecated @since @throws @exception @see @serial @serialField @serialData {@link}
- 사용자 정의 태그 및 주석

다음 태그의 대부분은 Java Language Specification, First Edition에 지정되어 있습니다. 또한 참조 자바 독 기준 페이지 .

태그 순서

다음 순서로 태그를 포함하십시오.

- `@author` (클래스 및 인터페이스 만, 필수)
- `@version`(클래스 및 인터페이스 만, 필수. 각주 1 참조)
- `@param` (메소드 및 생성자 만 해당)
- `@return` (방법 만)
- `@exception` (@throws는 Javadoc 1.2에 추가 된 동의어입니다.)
- `@see`
- `@since`
- `@serial` (또는 @serialField 또는 @serialData)
- `@deprecated`(API 지원 중단 방법 및시기 참조)

여러 태그 주문

문서 주석에 태그가 두 번 이상 나타날 때 다음 규칙을 사용합니다. 원하는 경우 여러 @see 태그와 같은 태그 그룹을 별표 하나가있는 빈 줄로 다른 태그와 구분할 수 있습니다.

여러 @author 태그는 시간순으로 나열되어야하며 클래스 작성자가 맨 위에 나열되어야합니다.

여러 @param 태그는 인수 선언 순서로 나열되어야합니다. 이렇게하면 목록을 선언에 시각적으로 더 쉽게 일치시킬 수 있습니다.

여러 @throws 태그 (@exception이라고도 함)는 예외 이름에 따라 알파벳순으로 나열되어야합니다.

여러 @see 태그는 자신의 인수가되는 것과 같은 순서로 약이다, 다음과 같이 주문해야 javadoc에 의해 검색 , 기본적으로 가까운에서 먼 액세스에 완전한, 다음 목록을 보여줍니다이 진행에서 최소 자격가. 메서드와 생성자는 "텔레 스코핑"순서로되어 있습니다. 즉, 먼저 "인자 없음"형식, "1 인수"형식, "2 인수"형식 등을 의미합니다. 두 번째 정렬 키가 필요한 경우 알파벳순으로 나열하거나 논리적으로 그룹화 할 수 있습니다.

```
@see #field
@see #Constructor(Type, Type...)
@see #Constructor(Type id, Type id...)
@see #method(Type, Type,...)
@see #method(Type id, Type, id...)
@see Class
@see Class#field
@see Class#Constructor(Type, Type...)
@see Class#Constructor(Type id, Type id)
@see Class#method(Type, Type,...)
@see Class#method(Type id, Type id,...)
@see package.Class
@see package.Class#field
@see package.Class#Constructor(Type, Type...)
@see package.Class#Constructor(Type id, Type id)
@see package.Class#method(Type, Type,...)
@see package.Class#method(Type id, Type, id)
@see package
```

필수 태그

@param 태그는 설명이 분명한 경우에도 모든 매개 변수에 대해 "필수"(관례 상)입니다. @return 태그는 `void`메서드 설명과 중복되는 경우에도 이외의 항목을 반환하는 모든 메서드에 필요합니다 . (가능하면 태그 주석에 사용할 중복되지 않는 (이상적으로는 더 구체적인) 것을 찾으십시오.)

이러한 원칙은 자동 검색 및 자동 처리를 촉진합니다. 종종 중복을 방지하려는 노력은 더욱 명확 해집니다.

태그 주석

다시 말씀 드리지만, 이러한 태그의 기본적인 사용은 Javadoc 참조 페이지 에 설명되어 있습니다. Java 소프트웨어는 일반적으로 다음 추가 지침을 사용하여 각 태그에 대한 주석을 작성합니다.

@author (참조 페이지)

@author 태그 하나, @author 태그를 여러 개 제공하거나 @author 태그를 제공하지 않을 수 있습니다. 새로운 API 개발이 열린 공동 노력 인 요즘 커뮤니티 프로세스에서 JSR은 패키지 수준에서 새 패키지의 작성자로 간주 될 수 있습니다. 예를 들어, 새 패키지 java.nio에는 패키지 수준에서 "@author JSR-51 Expert Group"이 있습니다. 그런 다음 개별 프로그래머를 클래스 수준에서 @author에게 할당 할 수 있습니다. 이 태그는 개요, 패키지 및 클래스 수준에서만 적용 할 수 있으므로 태그는 설계 또는 구현에 크게 기여한 사람에게만 적용되므로 일반적으로 기술 작성자는 포함하지 않습니다.

@author 태그는 API 사양을 생성 할 때 포함되지 않으므로 소스 코드를 보는 사람에게만 표시되므로 중요하지 않습니다. (버전 기록은 내부 목적으로 기여자를 결정하는 데 사용할 수도 있습니다.)

누군가 멤버 수준에서 @author를 추가해야한다고 강하게 느낀다면 새로운 1.4 옵션을 사용하여 javadoc을 실행하면됩니다 . `-tag`

```
-tag author:a:"Author:"
```

작성자를 알 수없는 경우 @author에 대한 인수로 "unscribed"를 사용하십시오. 여러 @author 태그의 순서도 참조하십시오 .

@version (참조 페이지)

@version 태그에 대한 인수에 대한 Java 소프트웨어 규칙은 SCCS 문자열 "% I %, % G %"이며 `1.39, 02/28/97`파일이 SCCS에서 체크 아웃 될 때 " "(mm / dd / yy) 와 같은 형식으로 변환됩니다 .

@param (참조 페이지)

@param 태그 뒤에는 매개 변수의 이름 (데이터 유형이 아님)과 매개 변수에 대한 설명이옵니다. 관례 적으로 설명의 첫 번째 명사는 매개 변수의 데이터 유형입니다. ("a", "an"및 "the"와 같은 기사는 명사 앞에 올 수 있습니다.) `int`데이터 유형이 일반적으로 생략되는 primitive에 대한 예외가 있습니다. 설명이 블록에 정렬되도록 이름과 설명 사이에 추가 공백을 삽입 할 수 있습니다. Javadoc 도구는 대시 하나를 삽입하므로 설명 앞에 대시 또는 기타 구두점을 삽입하면 안됩니다.

매개 변수 이름은 규칙에 따라 소문자입니다. 데이터 유형은 클래스가 아닌 객체를 나타 내기 위해 소문자로 시작합니다. 설명은 구문 (동사가 포함되지 않음)이면 소문자로 시작하고 문장이면 대문자로 시작합니다. 다른구나 문장이 뒤에 오는 경우에만 마침표로 구를 끝냅니다.

예:

```
* @param ch        the character to be tested
* @param observer  the image observer to be notified
```

`<code>...</code>`Javadoc 1.2 이상에서는 자동으로이 작업을 수행하므로 @param 태그 뒤에 매개 변수 이름을 괄호로 묶지 마십시오. (1.4부터는 이름에 HTML이 포함될 수 없습니다. Javadoc은 @param 이름을 서명에 나타나는 이름과 비교하고 차이가 있으면 경고를 내 보냅니다.)

주석 자체를 작성할 때는 일반적으로 구로 시작하여 필요한 경우 문장을 따라 가십시오.

- 문구를 쓸 때 대문자로 쓰지 말고 마침표로 끝나지 마십시오.

```
@param x  the x-coordinate, measured in pixels
```

- 문장 다음에 구를 쓸 때, 구를 대문자로 쓰지 말고 마침표로 끝내서 다음 문장의 시작과 구별하십시오.

```
@param x  the x-coordinate. Measured in pixels.
```

- 문장으로 시작하려면 대문자로하고 마침표로 끝냅니다.

```
@param x  Specifies the x-coordinate, measured in pixels.
```

- 여러 문장을 쓸 때 일반적인 문장 규칙을 따르십시오.

```
@param x  Specifies the x-coordinate. Measured in pixels.
```

여러 @param 태그의 순서도 참조하십시오 .

@return (참조 페이지)

void를 반환하는 메서드와 생성자에 대해서는 @return을 생략하십시오. 내용이 메소드 설명과 완전히 중복되는 경우에도 다른 모든 메소드에 포함하십시오. 명시적인 @return 태그를 사용하면 누군가가 반환 값을 더 쉽게 찾을 수 있습니다. 가능할 때마다 특수한 경우에 대한 반환 값을 제공하십시오 (예 : 범위를 벗어난 인수가 제공 될 때 반환되는 값 지정).

@param에서 사용한 것과 동일한 대문자 및 구두점을 사용하십시오.

@deprecated (참조 페이지)

첫 번째 문장의 @deprecated 설명은 최소한 API가 사용되지 않는시기와 대체 방법으로 사용할 내용을 사용자에게 알려야합니다. 요약 섹션과 색인에는 첫 번째 문장 만 표시됩니다. 후속 문장은 왜 그것이 더 이상 사용되지 않는지 설명 할 수 있습니다. 더 이상 사용되지 않는 API에 대한 설명을 생성 할 때 Javadoc 도구는 @deprecated 텍스트를 설명 앞으로 이동하여 기울임 꼴로 표시하고 그 앞에 굵은 경고 "Deprecated"를 표시합니다. `@see`태그 또는 (자바 독 1.1) `{@link}`태그 (자바 독 이상 또는 1.2)을 여분의 방법을 가리 포함한다 :

- Javadoc 1.2 이상의 경우 표준 형식은 `@deprecated`태그와 인라인 `{@link}`태그를 사용하는 것입니다. 그러면 원하는 위치에 인라인 링크가 생성됩니다. 예를 들면 :

```
/**
* @deprecated  As of JDK 1.1, replaced by
*              {@link #setBounds(int,int,int,int)}
*/
```

- Javadoc 1.1의 경우 표준 형식은 `@deprecated`및 `@see`태그 쌍을 작성하는 것 입니다. 예를 들면 :

```
/**
* @deprecated  As of JDK 1.1, replaced by
*
 setBounds
* @see #setBounds(int,int,int,int)
*/
```

멤버에 대체가없는 경우 @deprecated에 대한 인수는 "대체 없음"이어야합니다.

적절한 엔지니어에게 먼저 확인하지 않고 @deprecated 태그를 추가하지 마십시오. 실질적인 수정도 마찬가지로 먼저 확인해야합니다.

@since (참조 페이지)

Java 이름이 API 사양에 추가되었을 때 제품 버전을 지정합니다 (구현과 다른 경우). 예를 들어 패키지, 클래스, 인터페이스 또는 멤버가 버전 1.2에서 Java 2 Platform, Standard Edition, API 사양에 추가 된 경우 다음을 사용합니다.

```
/**
 * @since 1.2
 */
```

Javadoc 표준 doclet은 문자열 인수를 텍스트로 사용하여 "Since"부제목을 표시합니다. 이 부제목 `@since`은 소스 문서 주석에서 태그가 나타나는 위치에 해당하는 위치에서만 생성 된 텍스트에 나타납니다 (Javadoc 도구는 계층 구조 아래로 확장되지 않음).

(한때 규칙은 " `@since JDK1.2`"이었지만 이것은 Oracle JDK 또는 SDK에 특정한 것이 아니라 Java 플랫폼의 사양이기 때문에 "JDK"를 삭제했습니다.)

패키지가 도입되면 `@since`패키지 설명과 각 클래스에 태그를 지정하십시오 . (`@since`각 클래스에 태그를 추가하는 것은 기술적으로 필요하지 않지만 소스 코드에서 더 많은 가시성을 가능하게하는 우리의 관습입니다.) 재정의 태그가 없으면 태그의 값이 `@since`각 패키지의 클래스와 멤버에 적용됩니다.

클래스 (또는 인터페이스)가 도입되면 `@since`클래스 설명에 하나의 태그를 지정 `@since`하고 멤버 에는 태그를 지정 하지 마십시오. `@since`클래스보다 이후 버전에서 추가 된 멤버에만 태그를 추가하십시오 . 이렇게하면 `@since`태그 수가 최소화 됩니다.

멤버가 이후 릴리스에서 protected에서 public으로 `@since`변경되면 태그는 이제 하위 클래스가 아닌 모든 호출자가 사용할 수 있더라도 변경되지 않습니다.

@throws (**@exception** 이 원래 태그 임) (참조 페이지)

아래 그림과 같이 확인 된 예외 (throws 절에 선언 됨)와 호출자가 합리적으로 포착 할 수있는 확인되지 않은 예외에 대해 @throws 태그가 포함되어야합니다 `NullPointerException`. 오류는 예측할 수 없으므로 문서화해서는 안됩니다. 자세한 내용 은 @throws 태그를 사용하여 예외 문서화 를 참조하십시오 .

```
/**
* @throws IOException  If an input or output
*                      exception occurred
*/
public void f() throws IOException {
// body
}
```

예외 에 대한 자세한 내용은 *Java Language Specification, Second Edition* 의 예외 장 을 참조하십시오 . 또한 여러 @throws 태그의 순서를 참조하십시오 .

@see (참조 페이지)

여러 @see 태그의 순서도 참조하십시오 .

@연속물

@serialField

@serialData

(모두 Javadoc 1.2에 추가됨) (참조 페이지)

예제와 함께 이러한 태그를 사용하는 방법에 대한 정보는 "클래스에 대한 직렬화 가능한 필드 및 데이터 문서화" , Java 객체 직렬화 사양의 섹션 1.6을 참조하십시오 . 직렬화 된 양식 사양에 클래스를 포함하기위한 Oracle의 기준 도 참조하십시오 .

{@link} (Javadoc 1.2에 추가됨) (참조 페이지)

규칙은 경제적으로 인라인 링크 사용을 참고 하십시오 .

사용자 정의 태그 및 주석

어노테이션이 새로운 경우 소스 코드를 마크 업해야 할 때 어노테이션 또는 Javadoc 사용자 정의 태그를 사용할지 즉시 명확하지 않을 수 있습니다. 다음은 두 가지를 빠르게 비교 한 것입니다.

일반적으로 마크 업이 문서에 영향을 주거나 문서를 생성하려는 경우에는 아마도 javadoc 태그 여야합니다. 그렇지 않으면 주석이어야합니다.

- 태그-문서에 구조와 내용을 추가하는 방법으로 사용됩니다. 여러 줄 텍스트를 제공 할 수 있습니다. (사용자 정의 태그를 생성 하려면 -tag 또는 -taglet Javadoc 옵션을 사용하십시오 .) 태그는 프로그램 의미에 영향을주지 않아야합니다.
- 주석-프로그램 시맨틱에 직접적인 영향을주지 않지만 도구 및 라이브러리에서 프로그램을 처리하는 방식에 영향을 미치며, 이는 실행중인 프로그램의 시맨틱에 영향을 미칠 수 있습니다. 주석은 소스 파일, 클래스 파일에서 읽거나 런타임에 반영 될 수 있습니다. 한 줄의 텍스트를 제공 할 수 있습니다.

프로그램 의미와 문서에 모두 영향을 주어야하는 경우 주석과 태그가 모두 필요할 수 있습니다. 예를 들어, 이제 가이드라인 `@Deprecated`에서는 컴파일러 경고를 알리는 주석과 `@deprecated`주석 텍스트에 대한 태그를 사용할 것을 권장합니다.

기본 생성자 문서화

Java Language Specification, Second Edition의 8.8.7 절 에서는 기본 생성자를 설명합니다. 클래스에 생성자 선언이 포함되어 있지 않으면 매개 변수를 사용하지 않는 기본 생성자가 자동으로 제공됩니다. 인수없이 수퍼 클래스 생성자를 호출합니다. 생성자는 클래스와 동일한 액세스 권한을 갖습니다.

Javadoc 도구는 기본 생성자에 대한 문서를 생성합니다. 이러한 생성자를 문서화 할 때 기본 생성자는 문서 주석을 가질 수 없으므로 Javadoc은 설명을 공백으로 둡니다. 그러면 질문이 생깁니다. 기본 생성자에 대한 문서 주석을 어떻게 추가합니까? 간단한 대답은 불가능하다는 것입니다. 편리하게도 우리의 프로그래밍 규칙은 기본 생성자를 피하는 것입니다. (우리는 Javadoc 도구가 기본 생성자에 대한 기본 주석을 생성해야한다는 생각을 고려했지만 거부했습니다.)

좋은 프로그래밍 관행은 코드가 공용 API에서 기본 생성자를 사용해서는 안된다는 것을 지시합니다. **모든 생성자는 명시적이어야합니다.** 즉, 공용 및 보호 클래스의 모든 기본 생성자는 적절한 액세스 수정자를 사용하여 명시 적 생성자 선언으로 변환되어야합니다. 이 명시 적 선언은 문서 주석을 작성할 수있는 공간도 제공합니다.

이것이 좋은 프로그래밍 관행 인 이유는 디자인 엔지니어가 실제로 생성자의 액세스에 대해 결정을 내려야하기 때문에 명시 적 선언이 클래스가 실수로 인스턴스화되는 것을 방지하는 데 도움이되기 때문입니다. 우리는 퍼블릭 클래스가 인스턴스화되는 것을 원하지 않는 몇 가지 경우가 있었지만 프로그래머는 기본 생성자가 퍼블릭이라는 사실을 간과했습니다. 클래스가 제품의 릴리스 된 버전에서 실수로 인스턴스화 될 수 있도록 허용 된 경우 상위 호환성은 의도하지 않은 생성자가 향후 버전에서 유지되도록합니다. 이러한 불행한 상황에서는 생성자를 명시 적으로 지정하고 사용하지 않아야합니다 (사용 `@deprecated`).

명시 적 생성자를 만들 때 자동으로 생성 된 생성자의 선언 과 정확히 일치해야합니다 . 생성자를 논리적으로 보호해야하는 경우에도 호환성을 위해 자동으로 생성 된 생성자의 선언과 일치하도록 공개해야합니다. 그런 다음 적절한 문서 주석을 제공해야합니다. 종종 주석은 다음과 같이 간단해야합니다.

```
/**
* Sole constructor. (For invocation by subclass
* constructors, typically implicit.)
*/
protected AbstractMap() { }
```

@throws 태그로 예외 문서화

참고-태그 `@throws`와 `@exception`는 동의어입니다.

API 사양에서 예외 문서화

메서드에 대한 API 사양은 호출자와 구현 자 간의 계약입니다. Javadoc 생성 API 문서에는 예외에 대해이 계약을 지정하는 두 가지 방법, 즉 선언의 "throws"절과 `@throws`Javadoc 태그가 포함되어 있습니다. 이 가이드 라인은 `@throws`태그로 예외를 문서화하는 방법을 설명합니다 .

태그 던짐

`@throws`태그 의 목적은 프로그래머가 포착해야하는 예외 (선택된 예외의 경우) 또는 포착하려는 예외 (확인되지 않은 예외의 경우)를 나타내는 것입니다.

지침-문서화 할 예외

`@throws`태그 와 함께 다음 예외를 문서화하십시오 .

- **확인 된 모든 예외.**
  (Throws 절에서 선언해야합니다.)
- **호출자가 합리적으로 포착하고자 할 수있는 확인되지 않은 예외.**
  (Throws 절에 확인되지 않은 예외를 포함하는 것은 좋지 않은 프로그래밍 관행으로 간주됩니다.) 태그에
  이러한 예외를 문서화하는 것은 `@throws`아래 설명 된대로 API 디자이너의 판단에 달려 있습니다.

확인되지 않은 예외 문서화

일반적으로 메서드가 throw 할 수있는 검사되지 않은 예외를 문서화하는 것이 바람직합니다. 이렇게하면 호출자가 이러한 예외를 처리 할 수 있습니다 (필수는 아님). 예를 들어, 호출자가 구현에 따라 확인되지 않은 예외를 호출자의 내 보낸 추상화에 더 적합한 다른 예외로 "변환"할 수 있습니다.

호출이 발생시킬 수있는 확인되지 않은 모든 예외를 문서화했는지 보장 할 수있는 방법이 없기 때문에 프로그래머는 메서드가 발생하도록 문서화 된 예외 이외의 확인되지 않은 예외를 발생시킬 수 없다는 가정에 의존해서는 안됩니다. 즉, 메서드가 문서화되지 않은 확인되지 않은 예외를 throw 할 수 있다고 항상 가정해야합니다.

메서드가 해당 메서드의 현재 구현과 연결된 확인되지 않은 예외를 throw한다는 것을 문서화하는 것은 항상 적절하지 않습니다. 즉, 기본 구현과 독립적 인 예외를 문서화하십시오. 예를 들어, 인덱스를 사용하고 내부적으로 배열을 사용하는 메서드는 다른 구현에서 내부적으로 배열이 아닌 데이터 구조를 사용할 수 있으므로 를 throw하도록 문서화 되어서는 안됩니다`ArrayIndexOutOfBoundsException` . 그러나 일반적으로 이러한 메서드가
`IndexOutOfBoundsException`.

확인되지 않은 예외를 문서화하지 않으면 다른 구현에서 해당 예외를 throw하지 않아도됩니다. 예외를 적절하게 문서화하는 것은 한 번만 작성하고 어디서나 실행할 수있는 중요한 부분입니다.

확인 및 확인되지 않은 예외에 대한 배경

예외 확인 의 기본 개념 은 컴파일러가 컴파일시 예외가 try-catch 블록에서 제대로 포착되고 있는지 확인하는 것입니다.

다음과 같이 확인 된 예외와 확인되지 않은 예외를 식별 할 수 있습니다.

- 체크되지 않는 예외 클래스입니다 `RuntimeException`, `Error`자신의 서브 클래스.
- 다른 모든 예외 서브 클래스는 검사 된 예외입니다.

예외가 검사되는지 여부는 throws 절에 포함되는지 여부에 의해 정의되지 않습니다.

Throws 절의 배경

확인 된 예외는 메서드의 throws 절에 포함되어야합니다. 이것은 컴파일러가 검사 할 예외를 알기 위해 필요합니다. 예를 들어 (java.lang.Class에서) :

```
public static Class forName(String className)
throws ClassNotFoundException
```

부

관례 상 확인되지 않은 예외는 throws 절에 포함되지 않아야합니다. (이를 포함하는 것은 잘못된 프로그래밍 관행으로 간주됩니다. 컴파일러는이를 주석으로 처리하고 확인하지 않습니다.) 다음은 불량 코드입니다. 예외는 RuntimeException 이므로 `@throws`대신 태그에 문서화해야합니다 .

java.lang.Byte 소스 코드 :

```
public static Byte valueOf(String s, int radix)
throws NumberFormatException
```

부

참고는 것을 *Java* 언어 사양 도 절을 던졌습니다에 체크되지 않은 예외를 (같은 다음)를 보여줍니다. 사양에서 확인되지 않은 예외에 throws 절을 사용하는 것은이 예외가 호출자와 구현 자 간의 계약의 일부임을 나타내는 장치 일뿐입니다. 다음은 이에 대한 예입니다 (비교를 더 간단하게 만들기 위해 "최종"및 "동기화"가 제거됨).

`java.util.Vector` 소스 코드:

```
public Object elementAt(int index)
```

`java.util.Vector`Java 언어 사양, 1st Ed. (656 쪽) :

```
public Object elementAt(int index)
throws IndexOutOfBoundsException
```

부

패키지 수준 설명

Javadoc 1.2에서는 패키지 레벨 문서 주석을 사용할 수 있습니다. 각 패키지에는 Javadoc 도구가 생성하는 문서에 병합되는 자체 패키지 레벨 문서 주석 소스 파일이있을 수 있습니다. 이 파일은 이름이 `package.html`지정되며 모든 패키지에

대해 동일한 이름입니다. 이 파일은 모든 `*.java`파일 과 함께 소스 디렉토리에 보관됩니다 . (`packages.html`새 doc-files 소스 디렉토리에 파일을 넣지 마십시오. 해당 파일은 대상에만 복사되고 처리되지 않기 때문입니다.)

파일 에 package.html을 위한 패키지 레벨 소스 파일의 예는 `java.text`코드> 및 패키지는 summary.html에 Javadoc 툴에서 생성하는 파일이다.

Javadoc 도구 `package.html`는 다음 세 가지를 수행하여 처리 합니다.

대상 파일의 요약 테이블 아래에있는 내용 (`<body>`과 사이의 모든 항목)을 복사 `</body>`합니다 `package-summary.html`.
프로세스 어떤 `@see`, `@since`또는 `{@link}`존재의 Javadoc 태그입니다.
개요 요약 의 오른쪽 열에 첫 번째 문장을 복사합니다 .
package.html 소스 파일 용 템플릿

Oracle에서는 새 패키지 문서 주석 파일을 생성 할 때 다음 템플릿 인 패키지 수준 문서 주석 파일 용 빈 템플릿을 사용합니다. 여기에는 저작권 정보가 포함되어 있습니다. 분명히 다른 회사에서 온 경우 자신의 저작권 정보를 제공해야합니다. 엔지니어는이 전체 파일을 복사하고로 이름을 바꾸고 `package.html`해시 표시로 설정된 줄을 삭제합니다 `#####`.. 이러한 파일 하나는 소스 트리의 각 패키지 디렉토리로 이동해야합니다.

package.html 소스 파일의 내용

패키지 문서 주석은 프로그래머가 패키지를 사용하는 데 필요한 모든 것을 (직접 또는 링크를 통해) 제공해야합니다. 이것은 매우 중요한 문서입니다. 많은 시설 (단일 패키지에 있지만 단일 클래스에 있지 않은 시설)의 경우 프로그래머가 문서를 찾는 첫 번째 장소입니다. 패키지에서 제공하는 기능에 대한 짧고 읽기 쉬운 설명 (아래 소개 참조)과 세부 문서에 대한 포인터 또는 세부 문서 자체 중 적절한 것을 포함해야합니다. 적절한 것은 패키지에 따라 다릅니다. 더 큰 시스템 (예 : Corba의 37 개 패키지 중 하나)의 일부이거나 패키지에 대한 Framemaker 문서가 이미 존재하는 경우 포인터가 적합합니다.`java.math`).

요약하면, 패키지 문서 주석의 주요 목적은 패키지의 목적, 패키지를 이해하고 사용하는 데 필요한 개념적 프레임 워크, 패키지를 구성하는 클래스 간의 관계를 설명하는 것입니다. 크고 복잡한 패키지 (및 크고 복잡한 API의 일부인 패키지)의 경우 외부 아키텍처 문서에 대한 포인터가 보장됩니다.

다음은 패키지 수준 주석 파일을 작성할 때 사용해야하는 섹션과 제목입니다. Javadoc 도구는 요약 명령문으로 첫 번째 텍스트를 선택하므로 첫 번째 문장 앞에 표제가 없어야합니다.

- 첫 번째 문장을 패키지 요약으로 만드십시오. 예 : "자연어와 독립적 인 방식으로 텍스트, 날짜, 숫자 및 메시지를 처리하기위한 클래스 및 인터페이스를 제공합니다."
- 패키지에 포함 된 내용을 설명하고 용도를 명시하십시오.
- **패키지 사양**
  - **javadoc 생성 문서의 나머지 부분에 포함되지 않은이 패키지에 대한 패키지 전체 사양에 대한 설명이나 링크를 포함합니다.** 예를 들어, java.awt 패키지는 해당 패키지의 일반적인 동작이 운영 체제 (Windows, Solaris, Mac)에 따라 달라지는 방식을 설명 할 수 있습니다.
  - **javadoc 생성 파일에없는 어설 션이 포함 된 경우 문서 주석 (예 : FrameMaker 등) 외부에 작성된 모든 사양에 대한 링크를 포함 합니다.**

  주장은 규격에 부합하는 구현은 자바 플랫폼을 구현하기 위해 알아야 할 것 문입니다.

  따라서 Oracle에서이 섹션의 참조는 JCK (Java Compatibility Kit)에 매우 중요합니다. Java 호환성 키트에는 각 어설션을 확인하고 Java 호환으로 통과하는 항목을 확인하는 테스트가 포함되어 있습니다. "Returns an int"라는 문장은 어설 션입니다. 예는 주장이 아닙니다.

  엔지니어가 작성한 일부 "사양"에는 API 사양 (javadoc)에 아직 명시되지 않은 어설 션이 포함되어 있지 않습니다. API 사양에 대해 자세히 설명합니다. 이와 관련하여 이러한 문서는이 섹션에서 참조하지 말고 다음 섹션에서 참조해야합니다.

  - **특정 참조를 포함하십시오.** 참조 된 문서의 섹션 만 API 사양의 일부로 간주되어야하는 경우 해당 섹션 만 링크하거나 참조하고 다음 섹션의 나머지 문서를 참조해야합니다. 아이디어는 API 사양의 일부와 그렇지 않은 부분을 명

확하게 설명하는 것이므로 JCK 팀은 적절한 범위로 테스트를 작성할 수 있습니다. 이로 인해 일부 작성자는 문서를 분리하여 사양이 분리되도록 할 수도 있습니다.

- **관련 문서**
  - 개요, 자습서, 예제, 데모 및 가이드와 같은 사양 어설 션이 포함 되지 않은 문서에 대한 참조를 포함 합니다.
- **클래스 및 인터페이스 요약**
  - *[@category 태그를 구현할 때까지이 섹션 생략]*
- 클래스 및 인터페이스의 논리적 그룹화 설명
- `@see` 기타 패키지, 클래스 및 인터페이스

익명 내부 클래스 문서화

익명 내부 클래스는 Anonymous Class Declaration 에서 Java Language Specification, Second Edition에 정의되어 있습니다. Javadoc 도구는 익명 클래스를 직접 문서화하지 않습니다. 즉, 해당 선언 및 문서 주석이 무시됩니다. 익명 클래스를 문서화하려는 경우이를 수행하는 적절한 방법은 외부 클래스 또는 밀접하게 연관된 다른 클래스의 문서 주석에있는 것입니다.

예를 `TreeSelectionListener`들어이 클래스의 사용자가 재정의 할 수 `makeTree`있는 `JTree`객체 를 반환 하는 메서드에 익명 내부 클래스 가있는 경우 반환 된 메서드 `JTree`에 `TreeSelectionListener`첨부 된 메서드 주석을 문서화 할 수 있습니다.

```
/**
* The method used for creating the tree. Any structural
* modifications to the display of the Jtree should be done
* by overriding this method.
* <p>
* This method adds an anonymous TreeSelectionListener to
* the returned JTree.  Upon receiving TreeSelectionEvents,
* this listener calls refresh with the selected node as a
* parameter.
*/
public JTree makeTree(AreaInfo ai){
}
```

이미지 포함

이 섹션에서는 소스 코드에서 직접 사용하는 이미지가 아니라 문서 주석에 사용 된 이미지를 다룹니다.

참고 : Javadoc 버전 1.0 및 1.1에 필요한 글 머리 기호 및 제목 이미지는 JDK 다운로드 번들의 이미지 디렉토리에 `jdk1.1/docs/api/images/`있습니다. 이러한 이미지는 1.2부터 더 이상 필요하지 않습니다.

Javadoc 1.2 이전에는 Javadoc 도구가 이미지를 대상 디렉토리에 복사하지 않았습니다. 수동으로 또는 스크립트를 사용하여 별도의 작업을 수행해야했습니다. Javadoc 1.2는 소스 트리 (패키지 당 하나씩)와 그 내용에서 "doc-files"라는 이름의 디렉토리를 찾아 대상 디렉토리에 복사합니다. (1.2 및 1.3에 대해 얕은 복사를 수행하고 1.4 이상에 대해 딥 복사를 수행합니다.) 따라서 Javadoc 도구에서 달리 처리하지 않는 이미지 (GIF, JPEG 등) 또는 기타 파일을이 디렉토리에 넣을 수 있습니다.

다음은 문서 주석에 이미지를 포함하기위한 규칙에 대한 Java 소프트웨어 제안입니다. 마스터 이미지는 소스 트리에 있습니다. Javadoc 도구가 표준 doclet으로 실행되면 해당 파일을 대상 HTML 디렉토리로 복사합니다.

소스 트리의 이미지

- **소스 트리에있는 문서 이미지의 이름 지정** 이름 GIF 이미지 - `<class>-1.gif`같은 클래스에서 연속 된 이미지의 정수를 증가. 예:
  `Button-1.gif`

- **소스 트리에서 문서 이미지의 위치** - "doc-files"라는 디렉토리에 문서 이미지를 넣습니다. 이 디렉토리는 소스 파일이 있는 동일한 패키지 디렉토리에 있어야합니다. ("doc-files"라는 이름은 GUI에 표시되는 비트 맵과 같이 소스 코드 자체에서 사용하는 이미지와는 별개의 문서로 구별됩니다.) 예 : 버튼의 스크린 샷 `Button-1.gif`이 클래스 주석에 포함될 수 있습니다. 버튼 클래스. 버튼 소스 파일과 이미지는 다음 위치에 있습니다.
  `java/awt/Button.java` (소스 파일)

  `java/awt/doc-files/Button-1.gif` (이미지 파일)

HTML 대상의 이미지

- **HTML 대상에서 문서 이미지 이름 지정** -이미지는 소스 트리에있는 것과 동일한 이름을 갖습니다. 예:
  `Button-1.gif`

- **HTML 대상에서 문서 이미지의 위치**
  - Javadoc 1.2와 같은 계층 적 파일 출력의 경우 디렉토리는 "doc-files"라는 패키지 디렉토리에 있습니다. 예를 들면 :
    `api/java/awt/doc-files/Button-1.gif`
  - Javadoc 1.1과 같은 플랫 파일 출력의 경우 디렉토리는 패키지 디렉토리에 있으며 "images- <package>"라는 이름이 지정됩니다. 예를 들면 :

```
api/images-java.awt/
api/images-java.awt.swing/
```

문서 주석의 예

```
/**
* Graphics is the abstract base class for all graphics contexts
* which allow an application to draw onto components realized on
* various devices or onto off-screen images.
* A Graphics object encapsulates the state information needed
* for the various rendering operations that Java supports.  This
* state information includes:
* <ul>
* <li>The Component to draw on
* <li>A translation origin for rendering and clipping coordinates
* <li>The current clip
* <li>The current color
* <li>The current font
* <li>The current logical pixel operation function (XOR or Paint)
* <li>The current XOR alternation color
*     (see <a href="#setXORMode">setXORMode</a>)
* </ul>
* <p>
* Coordinates are infinitely thin and lie between the pixels of the
* output device.
* Operations which draw the outline of a figure operate by traversing
```

```
* along the infinitely thin path with a pixel-sized pen that hangs
* down and to the right of the anchor point on the path.
* Operations which fill a figure operate by filling the interior
* of the infinitely thin path.
* Operations which render horizontal text render the ascending
* portion of the characters entirely above the baseline coordinate.
* <p>
* Some important points to consider are that drawing a figure that
* covers a given rectangle will occupy one extra row of pixels on
* the right and bottom edges compared to filling a figure that is
* bounded by that same rectangle.
* Also, drawing a horizontal line along the same y coordinate as
* the baseline of a line of text will draw the line entirely below
* the text except for any descenders.
* Both of these properties are due to the pen hanging down and to
* the right from the path that it traverses.
* <p>
* All coordinates which appear as arguments to the methods of this
* Graphics object are considered relative to the translation origin
* of this Graphics object prior to the invocation of the method.
* All rendering operations modify only pixels which lie within the
* area bounded by both the current clip of the graphics context
* and the extents of the Component used to create the Graphics object.
*
* @author      Sami Shaio
* @author      Arthur van Hoff
* @version     %I%, %G%
* @since       1.0
*/
public abstract class Graphics {

/**
* Draws as much of the specified image as is currently available
* with its northwest corner at the specified coordinate (x, y).
* This method will return immediately in all cases, even if the
* entire image has not yet been scaled, dithered and converted
* for the current output device.
* <p>
* If the current output representation is not yet complete then
* the method will return false and the indicated
* {@link ImageObserver} object will be notified as the
* conversion process progresses.
*
* @param img       the image to be drawn
* @param x         the x-coordinate of the northwest corner
*                  of the destination rectangle in pixels
* @param y         the y-coordinate of the northwest corner
*                  of the destination rectangle in pixels
* @param observer  the image observer to be notified as more
*                  of the image is converted.  May be
*                  <code>null</code>
* @return          <code>true</code> if the image is completely
*                  loaded and was painted successfully;
*                  <code>false</code> otherwise.
* @see             Image
* @see             ImageObserver
* @since           1.0
*/
public abstract boolean drawImage(Image img, int x, int y,
ImageObserver observer);

/**
* Dispose of the system resources used by this graphics context.
* The Graphics context cannot be used after being disposed of.
```

```
* While the finalization process of the garbage collector will
* also dispose of the same system resources, due to the number
* of Graphics objects that can be created in short time frames
* it is preferable to manually free the associated resources
* using this method rather than to rely on a finalization
* process which may not happen for a long period of time.
* <p>
* Graphics objects which are provided as arguments to the paint
* and update methods of Components are automatically disposed
* by the system when those methods return.  Programmers should,
* for efficiency, call the dispose method when finished using
* a Graphics object only if it was created directly from a
* Component or another Graphics object.
*
* @see       #create(int, int, int, int)
* @see       #finalize()
* @see       Component#getGraphics()
* @see       Component#paint(Graphics)
* @see       Component#update(Graphics)
* @since     1.0
*/
public abstract void dispose();

/**
* Disposes of this graphics context once it is no longer
* referenced.
*
* @see       #dispose()
* @since     1.0
*/
public void finalize() {
dispose();
}
}
```

둥근 따옴표 문제 해결 (Microsoft Word)

문제 -PC의 Microsoft Word와 같이 둥근 (직선이 아닌) 작은 따옴표 및 큰 따옴표로 기본 설정되는 편집기에서 작업하는 경우 문제가 발생합니다. 일부 브라우저 (예 : Unix Netscape)에 표시 될 때 따옴표가 사라집니다. 따라서 "디스플레이의 특성"과 같은 문구는 "디스플레이 특성"이됩니다.

잘못된 문자는 다음과 같습니다.

- 146-오른쪽 작은 따옴표
- 147-왼쪽 큰 따옴표
- 148-오른쪽 큰 따옴표

대신 사용해야하는 것은 다음과 같습니다.

- 39-곧은 작은 따옴표
- 34-곧은 따옴표

예방 솔루션 - "불법"인용문이 발생한 이유는 기본 Word 옵션이 " '직선 인용문'을 '스마트 인용문'으로 변경"이기 때문입니다. 이 기능을 끄면 입력 할 때 적절한 곧은 따옴표가 표시됩니다.

곱슬 따옴표 수정 -Microsoft Word에는 여러 저장 옵션이 있습니다. "텍스트로만 저장"을 사용하여 따옴표를 곧은 따옴표로 다시 변경하십시오. 올바른 옵션을 사용해야합니다.

- "줄 바꿈이있는 텍스트로만 저장"-각 줄 끝에 공백을 삽입하고 둥근 따옴표를 유지합니다.
- "텍스트로만 저장"-각 줄 끝에 공백을 삽입하지 않고 둥근 따옴표를 곧은 따옴표로 변경합니다.

각주

[1] Java Software에서는 SCCS 버전으로 @version을 사용합니다. 자세한 내용은 "man sccs-get"을 참조하십시오. 합의는 다음과 같습니다.

% I %는 파일을 편집하고 삭제할 때마다 증가합니다.

% G %는 날짜 mm / dd / yy입니다.

파일을 만들 때 % I %는 1.1로 설정됩니다. 편집하고 삭제하면 1.2로 증가합니다.

일부 개발자는 날짜가 너무 헷갈 리면 % G % (그리고 그렇게하고 있음)를 생략합니다. 예를 들어, % G %가 3 월 4 일에 생성하는 96 년 3 월 4 일은 유나이티드 외부의 사람들이 해석합니다. 4 월 3 일을 의미하는 주. 일부 개발자는 (하루에 여러 번 체크인하기 때문에) 더 정밀한 해결을 원하는 경우에만 시간 % U %를 포함합니다.

가장 명확한 숫자 날짜 형식은 ISO 8601 및 다른 곳 (예 : http://www.cl.cam.ac.uk/~)에 제안 된대로 yyyy-mm-dd와 같이 연도를 먼저 형식으로 지정하는 것입니다. mgk25 / iso-time.html), 이러한 향상은 SCCS에서 가져와야합니다.