Java Code Conventions, Packaging & Delivery

including

Topics:

- Indentation / Naming / Comments / Class Definition
- JavaDocs / Packaging / Making JARs

Chapter 17 – “Head First Java” book
Section 5.12 – “Introduction to Java Programming” book
Chapter 7 – “Java in a Nutshell” book

Java Code Conventions: why?

- To produce programs that have *good style*.
- Improve *readability*: by author, by others.
- Design for *reusability*: can be reused later in another programs.
- Good *appearance*.
- *Maintenance*: reduce cost.
- Clean and well packaged as a product.
- *How to achieve this?*
  - Indentation
  - Naming
  - Comments
  - Source file organisation

Indentation

- The *blank space(s)* between a margin and the beginning of an indented line.
- Emphasises program structure.
- Unit: 4 spaces (or just *be consistent*).
- Indent every level:
  - when a new set of curly braces *or*
  - (a block) occurs.

Tip: Use IDE to format your source code and this becomes a very simple task.

Eclipse: Source → Format
NetBeans: Source → Reformat Code

Naming: class, method, variable, package names (1/3)

- To be a valid name:
  - Made up of letters, digits and underscore (_).
  - Start with a letter.
  - Can not be Java keywords (e.g. `char`, `transient`, …).
- To be a good name:
  - Simple.
  - Meaningful.

- Class names:
  - nouns
  - mixed case
  - capitalise 1st letter of each internal word
  - use whole words

```
public class Point {
  // ...
}
```

```
Other good names:
Diary
FileProcessor
CounterGUI
BlackJackGame
```

Naming: class, method, variable, package names (2/3)

- Variable names:
  - short
  - mixed case
  - 1st letter lowercase, and 1st letter of each internal word capitalised

```
Good names:
accountNo
accountName
balance
```

```
Common names for
temporary variables: i, j, k, m
integers: n
characters: c, d, e
```

- Constant name:
  - all uppercase
  - words separated by underscores ("_")
  - `final`

```
Other examples:
LIMIT
MAX_LENGTH
MIN_VALUE
```

```
final double PI = 3.1415926;
```

Naming: class, method, variable, package names (3/3)

- Method name:
  - verb
  - 1st letter lowercase, and 1st letter of each internal word capitalised

```
Examples:
reverse
changeCase
draw
deal
writeToFile
```

- Package name:
  - all-lowercase
  - use top-level domain name
    - `.com`, `.org`, `.gov`, `.net`, …

```
Examples:
cardgame
pontoon
carpark
cdplayer
```

Comments

- Documentation comments:
  `/** ... */` to describe the specification
  - all classes
  - at least all service methods
  - will be written in Java doc

Write comments!!!

- Implementation comments:
  `/* ... */` or `//` to comment out code or comment about the particular implementation.
- Very important when *generating Javadocs*.
- Write comments on top of a block of code.

Source File Organisation

```
/*
 * Beginning comments: File name, version, date, copyright etc ...
 */
package packagename;

import packagename.className;
import packagename.className;
 /**
  * Class documentation comments
  */
public class ClassName {
  static variables (1.public, 2.protected, 3.package level, 4.private)
  instance variables (1.public, 2.protected, 3.package level, 4.private)
  constructors (1.default constructor 2.user-defined constructors)
  methods (write documentation comments for each method)
           1.accessor methods
           2.service and support methods (grouped by functionality)
           3.toString
           4.main
}
```

Code example:
http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-137946.html#186

Code Conventions

- Code Conventions for the Java Programming Language: http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html
- The Java Language and Virtual Machine Specifications: http://docs.oracle.com/javase/specs/
- EBU4201 course website in QMplus, under the WRITING AND DEBUGGING PROGRAMS topic.

Javadocs (Revision)

- What is Javadoc:
  - A *tool for generating API documentation* in HTML format from documentation comments in source code.
  - As seen in Java standard library.

- How to generate Javadocs:
  - Command line:

    ```
    javadoc [options] [packagenames] [source files] [@files]
    ```

  - IDE:
    - Eclipse: Project → generate javadoc
    - NetBeans: Build → generate javadoc

Javadocs: Package level (Revision)

java.util (Java Platform SE

docs.oracle.com/javase/8/docs/api/index.html?java/util/package-summary.html

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Java™ Platform Standard Ed. 8

All Classes All Pro

Packages

All Classes

AbstractAction
AbstractAnnotationVal
AbstractAnnotationVal
AbstractAnnotationVal
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserI
AbstractDocument
AbstractDocument.Att
AbstractDocument.Co
AbstractDocument.Ele
AbstractElementVisito
AbstractElementVisito

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

See: Description

Interface Summary

Interface	Description
Collection<E>	The root interface in th *collection hierarchy.*
Comparator<T>	A comparison function, whic

Javadocs: Class level (Revision)

Scanner (Java Platform SE

docs.oracle.com/javase/8/docs/api/index.html?java/util/package-summary.html

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

java.time.zone
java.util
java.util.concurrent
java.util.concurrent.at
java.util.concurrent.lo
java.util.function
java.util.jar

SAXParseException
SAXParser
SAXParserFactory
SAXResult
SAXSource
SAXTransformerFactor
Scanner
ScatteringByteChanne
ScheduledExecutorSe
ScheduledFuture
ScheduledThreadPoolE
Schema
SchemaFactory
SchemaFactoryConfigu
SchemaFactoryLoader
SchemaOutputResolve
SchemaViolationExcep
ScriptContext

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.util

Class Scanner

java.lang.Object
    java.util.Scanner

All Implemented Interfaces:

Closeable, AutoCloseable, Iterator<String>

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

A simple text scanner which can parse primitive types and strings using regular expressions.

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods

Javadocs: method level (Revision)

Scanner (Java Platform SE

docs.oracle.com/javase/8/docs/api/index.html?java/util/package-summary.html

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

java.time.zone
java.util
java.util.concurrent
java.util.concurrent.at
java.util.concurrent.lo
java.util.function
java.util.jar

SAXParseException
SAXParser
SAXParserFactory
SAXResult
SAXSource
SAXTransformerFactor
Scanner
ScatteringByteChanne
ScheduledExecutorSe
ScheduledFuture
ScheduledThreadPoolE
Schema
SchemaFactory
SchemaFactoryConfigu
SchemaFactoryLoader
SchemaOutput
SchemaViolati
ScriptContext

nextFloat

```
public float nextFloat()
```

Scans the next token of the input as a `float`. This method will throw `InputMismatchException` if the next token cannot be translated into a valid float value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the *Float* regular expression defined above then the token is converted into a `float` value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via `Character.digit`, prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to `Float.parseFloat`. If the token matches the localized NaN or infinity strings, then either "Nan" or "Infinity" is passed to `Float.parseFloat` as appropriate.

Returns:

the float scanned from the input

Throws:

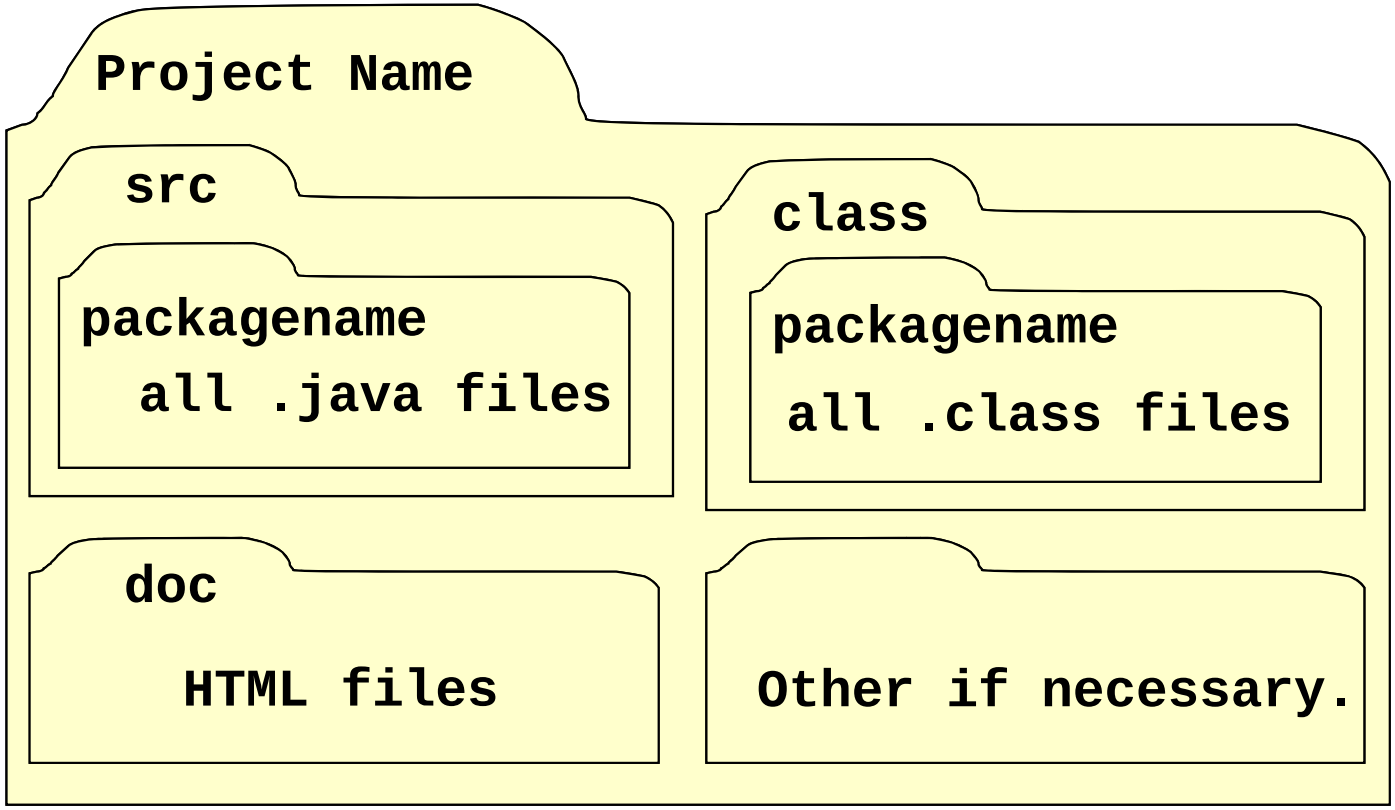
`InputMismatchException` - if the next token does not match the *Float* regular expression, or is out of range

Further Reading: "How to Write Doc Comments for the Javadoc Tool" at http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html.

Packaging

- Package your code: add package name.
- Separate source files and class files.
- Generate Javadoc.

- Use IDE to manage your files:
  - Separate source and class.
- Finally, deliver it!
  - Make a ZIP of your top project folder. *OR*
  - Make a JAR.



Code Delivery

- JAR:
  - JavaARchive
  - Executable
  - JAR files are packaged with the ZIP file format
- Benefits of JARs:
  - data compression
  - archiving
  - decompression
  - archive unpacking
- Making JAR from an IDE:
  - Eclipse: right click your project, export → Java → JAR
  - Netbeans: right click your project → build
- May include ***src***, ***class*** and ***doc***.

Further reading: "Packaging Programs in JAR Files" at http://docs.oracle.com/javase/tutorial/deployment/jar/index.html.

Example using `jar` commands

Making a JAR from the command line:

- Create → `jar cf jar-file input-file(s)`
- View the contents → `jar tf jar-file`
- Extract the contents → `jar xf jar-file`
- Run application → `java -jar app.jar`

CMD

```
C:\coursework\ELB2222>cd classes

C:\coursework\ELB2222\classes>jar cvf myjar.jar question1/Answer1.class question
1/Answer2.class question2/Answer1.class
added manifest
adding: question1/Answer1.class(in = 458) (out= 307)(deflated 32%)
adding: question1/Answer2.class(in = 458) (out= 307)(deflated 32%)
adding: question2/Answer1.class(in = 458) (out= 307)(deflated 32%)

C:\coursework\ELB2222\classes>dir
 Volume in drive C has no label.
 Volume Serial Number is 116F-B317

 Directory of C:\coursework\ELB2222\classes

23/10/2006  02:39    <DIR>          .
23/10/2006  02:39    <DIR>          ..
23/10/2006  02:39             1,680 myjar.jar
23/10/2006  01:44    <DIR>          question1
23/10/2006  01:44    <DIR>          question2
               1 File(s)          1,680 bytes
               4 Dir(s)  16,286,064,640 bytes free

C:\coursework\ELB2222\classes>
```