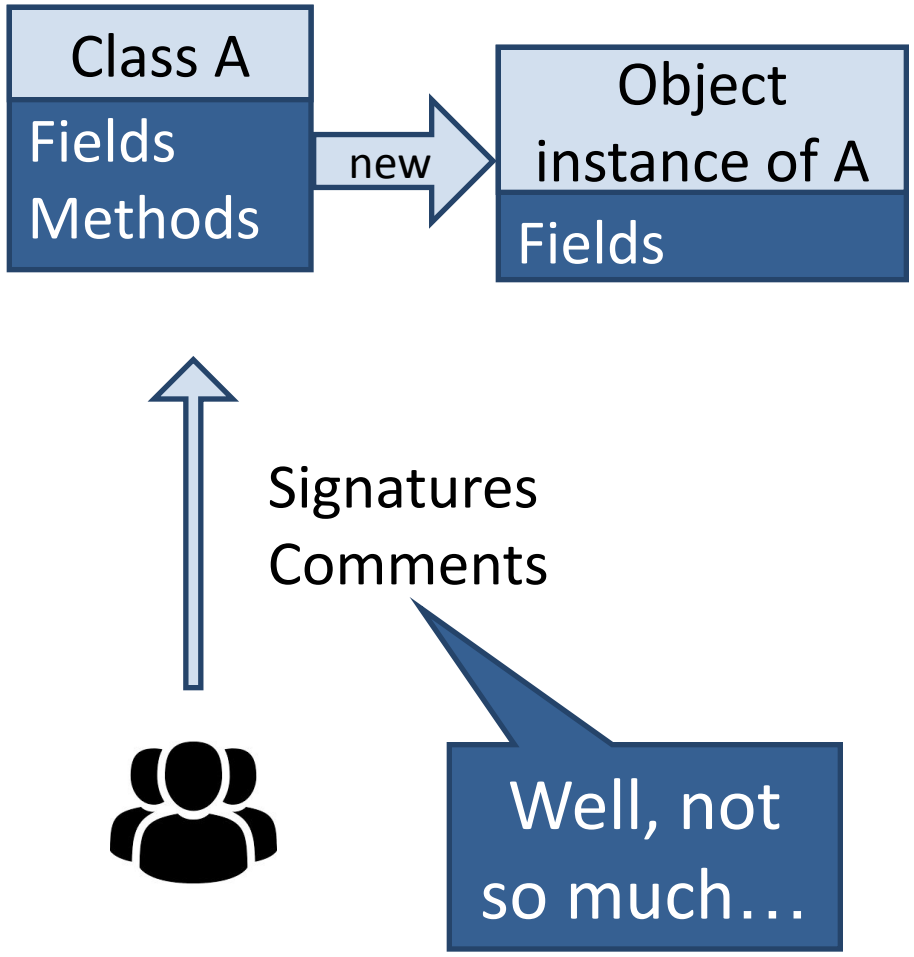
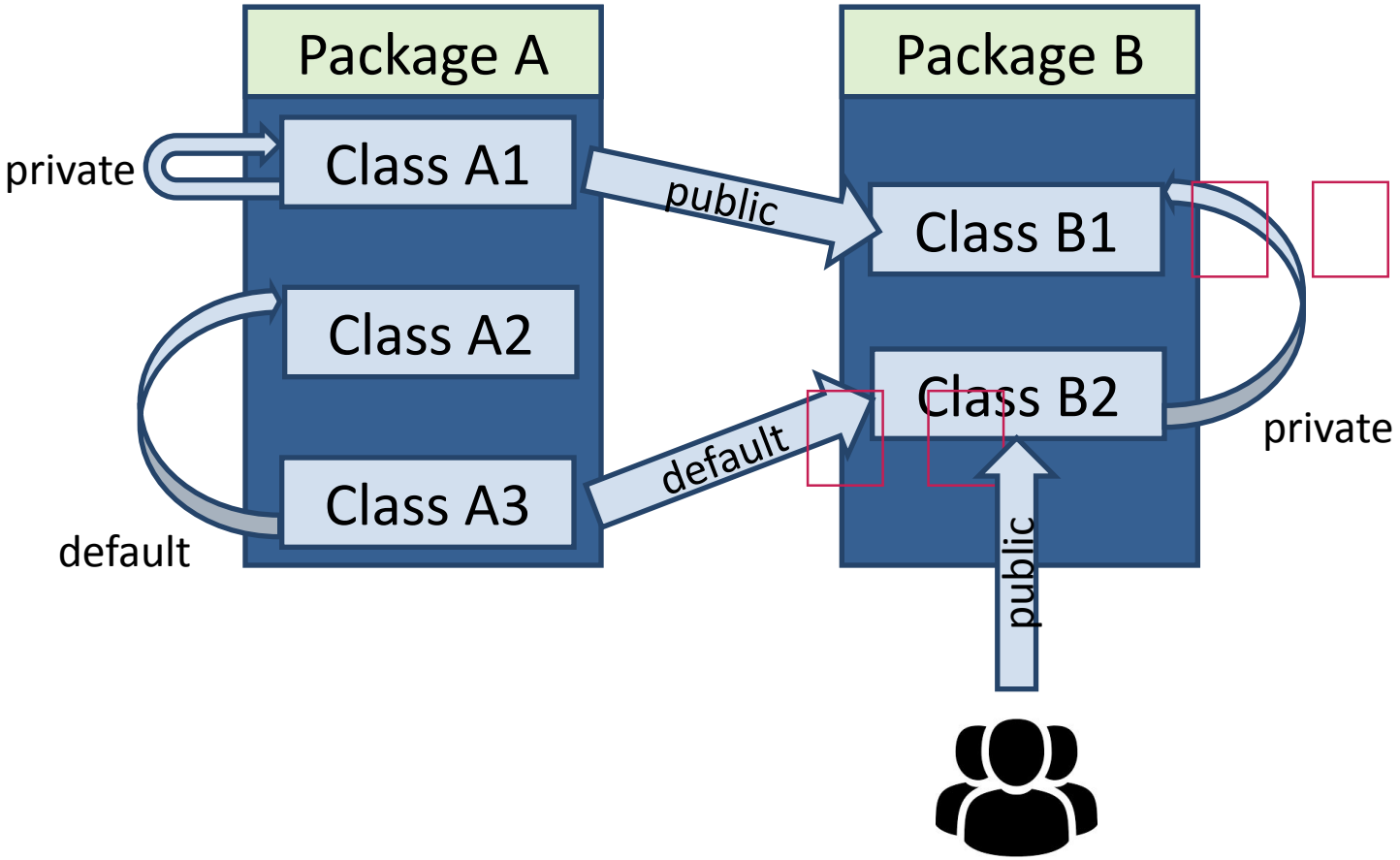
Summary


Class A
Fields
Methods
new
Object instance of A
Fields
Signatures
Comments
Well, not so much…


Package A
Package B
Class A1
Class A2
Class A3
Class B1
Class B2
private
public
default
default
private
public

Ca' Foscari
University
of Venice

Documentation and javadoc

Object oriented programming, module 1

Pietro Ferrara

pietro.ferrara@unive.it

Using capsules

- We have seen how OOPLs allow to build up software capsules
- Each capsule has a clear application programming interface (API)
- Usually, in Java capsules are delivered as libraries
  - A set of classes (belonging to some packages) packaged together
  - Delivered as a jar file
    - Zipped file containing several .class files with a clear structure
    - … and some other information (XML description, properties, etc..)
- Documentation is an essential part of capsules

jar command

- JDK ships a jar command that allows to create jar files

*The jar command is a general-purpose **archiving and compression tool**, based on the ZIP and ZLIB compression formats. Initially, the jar command was designed to **package Java applets** (not supported since JDK 11) or applications; however, beginning with JDK 9, users can use the jar command to create **modular JARs**.*

https://docs.oracle.com/en/java/javase/14/docs/specs/man/jar.html

- IDEs provide facilities to create jar files

jar files

- jar file can be executables
  - Specify what class contains the main method when packing it
- It contains a META-INF/MANIFEST.MF file

```
Manifest-Version: 1.0
Specification-Title: Java Platform API Specification
Specification-Version: 11
Specification-Vendor: Oracle Corporation
Implementation-Title: Java Runtime Environment
Implementation-Version: 11.0.6
Implementation-Vendor: Oracle Corporation
Created-By: 10 (Oracle Corporation)
```

- https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html

Ca' Foscari University of Venice

Abstraction and interfaces

- A class defines a contract specifying the interface of the objects
  - Method signature represents the structure
  - Method semantics (meaning) needs to be documented externally
- This allows to abstract away the internal implementation

From Lecture 2, side 14

```
class Car {
  //Add the given amount to the fuel tank
  void refuel(double amount) {…}
  //Increment the speed
  void accelerate(double a) {…}
  //Stop the car
  void fullBreak() {…}
}
```

Comments

- We all know that comments are important, isn't it?
  - And we all properly comment our code, isn't it?
- However, we need to distinguish among 2 types of comments
  - Source code comments explain what a part of the code computes
    - single line (starting with //)
    - multi line (starting with /* and ending with */)
  - Documentation (javadoc) comments explain the API of the library
    - Start with /** and end with */
- Javadoc command then generates HTML documentation pages
  - And they are used by IDEs for showing documentation

Ca' Foscari University of Venice

Example (java.lang.Arrays)

Internal comments

External documentation

```
/*
 * Sorting methods. Note that all public "sort" methods take the
 * same form: Performing argument checks if necessary, and then
 * expanding arguments into those required for the internal
 * implementation methods residing in other package-private
 * classes (except for legacyMergeSort, included in this class).
 */
/**
 * Sorts the specified array into ascending numerical order.
 *
 * <p>Implementation note: The sorting algorithm is a Dual-Pivot Quicksort
 * by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm
 * offers O(n log(n)) performance on many data sets that cause other
 * quicksorts to degrade to quadratic performance, and is typically
 * faster than traditional (one-pivot) Quicksort implementations.
 *
 * @param a the array to be sorted
 */
public static void sort(int[] a) {
    DualPivotQuicksort.sort(a, 0, a.length - 1, null, 0, 0);
}
```

javadoc

- Standard to document the APIs of Java libraries
- A section documents the element whose declaration follows:
  - Class, fields, methods
- Remember that objects are instances of classes, contain data in fields, provide functionalities through methods
  - Provide full documentation of the API
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html#sort(int%5B%5D)

javadoc elements

- Javadoc comments can contain HTML tags
- In addition, javadoc supports some internal tags
  - @literal <text>: print text exactly as it is
  - @code <code>: like @literal + source code formatting
  - @link <element>: adds a link to the element (class, field, method)
  - @see <element>: adds a link to the element in the See also section
- javadoc command produces some HTML pages

Documenting a class

- The javadoc section of a class usually contains:
  - @author: the author that wrote the class
  - @version: the version of the class
  - @since: the version when the class was added for the first time

```
/**
 * This class contains various methods for manipulating arrays (such as
 * sorting and searching). This class also contains a static factory
 * that allows arrays to be viewed as lists.
 *
 * @author Josh Bloch
 * @author Neal Gafter
 * @author John Rose
 * @since  1.2
 */
```

@version often does not make sense
Already in the manifest
Should be updated at each release

Ca' Foscari University of Venice

Documenting a method

- The interface of a method consists of
  - Return value (tag @return)
  - Parameters (tag @param <name>)
  - The exceptions it throws (tag @throws, will see it later during the course)
  - If it has been deprecated (tag @deprecated, will be discussed with libraries)

```
/**
 * Finds and returns the index of the first mismatch between two
 * {@code double} arrays, otherwise return -1 if no mismatch is found.
 * (…)
 * @param a the first array to be tested for a mismatch
 * @param b the second array to be tested for a mismatch
 * @return the index of the first mismatch between the two arrays,
 *         otherwise {@code -1}.
 * @throws NullPointerException
 *         if either array is {@code null}
 * @since 9
 */
public static int mismatch(double[] a, double[] b)
```

Documenting a field

- Nothing special, just the plain description

```
/**
   * The minimum array length below which a parallel sorting
   * algorithm will not further partition the sorting task. Using
   * smaller sizes typically results in memory contention across
   * tasks that makes parallel speedups unlikely.
   */
  private static final int MIN_ARRAY_SORT_GRAN = 1 << 13;
```

Ca' Foscari University of Venice

What should be documented with javadoc?

- Public methods/fields/classes: definitely yes!
- Private methods/fields/classes: usually no!
- Protected/default methods/fields/classes: it depends…
- https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html#CHDCHFEB
  - Options to define at what accessibility level we want to stop the API documentation (public, protected, package, private)

Some examples of javadoc documents

- Java 11 API (java.base):
  - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/module-summary.html
- Apache Commong Lang:
  - https://commons.apache.org/proper/commons-lang/apidocs/index.html

Doxygen

- https://www.doxygen.nl/

Doxygen is the de facto standard tool for generating documentation from ***annotated C++ sources****, but it also* ***supports other popular programming languages*** *such as C, Objective-C, C#, PHP,* ***Java****, Python, IDL (Corba, Microsoft, and UNO/OpenOffice flavors), Fortran, VHDL and to some extent D.*

- Generate documentation from annotated code (e.g., Javadoc)
- Generate other documentation from other documents
  - https://www.doxygen.nl/manual/starting.html
- Input and output can have various formats (Latex, pdf, …)

Ca' Foscari University of Venice

https://stackoverflow.com/questions/225447/doxygen-vs-javadoc

Doxygen has a number of features that JavaDoc does not offer, e.g. the class diagrams for the hierarchies and the cooperation context, more summary pages, optional source-code browsing (cross-linked with the documentation), additional tag support such as @todo on a separate page and it can generate output in TeX and PDF format.It also allows a lot of visual customization.

Since Doxygen supports the standard JavaDoc tags you can run Doxygen on any source code with JavaDoc comments on it. It often can even make sense to run on source code without JavaDoc since the diagrams and source code browsing can help understanding code even without the documentation. And since the JavaDoc tool ignores unknown tags you can even use additional Doxygen tags without breaking JavaDoc generation.

Having said all this I must admit that I haven't used Doxygen for a long time. I tend to rely heavily on my IDE nowadays to provide the same visualization and I usually don't read JavaDoc as HTML pages but import the source files into my IDE so it can generate JavaDoc flyouts and I can jump to the definitions. That's even more powerful than what Doxygen has to offer. If you want to have documentation outside the IDE and are happy to run non-Java tooling then Doxygen is worth a try since it doesn't require any change to your Java code.

Share Improve this answer Follow

answered Nov 27 '08 at 12:51

Peter Becker

8,535 ● 7 ● 39 ● 60

1. Doxygen does something more than Javadoc
2. It supports and uses Javadoc comments
3. It produces documentation outside the source code
4. It is partially/not integrated in the IDE

Materials

- Lecture notes: Chapter 5
- Arnold et al.: Chapter 19
- Official Oracle documentation
  - "How to Write Doc Comments for the Javadoc Tool", https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
  - Documentation of the Javadoc tool: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html