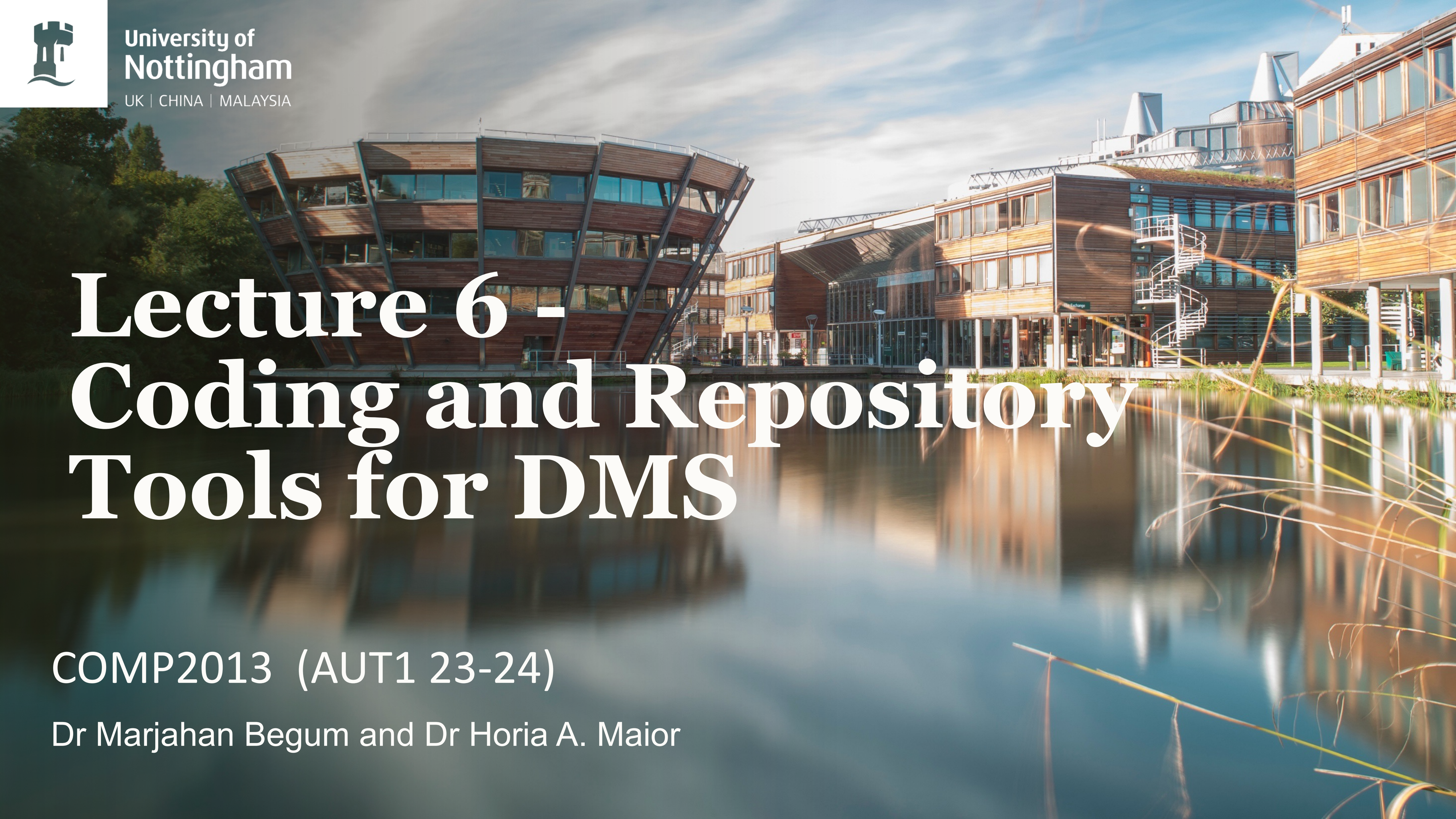


University of Nottingham
UK | CHINA | MALAYSIA

Lecture 6 - Coding and Repository Tools for DMS

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior

Register your attendance

COMP2013: Developing Maintainable Software
Week 7 – 4:00pm Monday – 06 November 2023

valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

Overview

- Module Feedback
- Version Control
- Setting up Git with Intellij
- Coding convention
- Javadoc

COMP2013: Developing Maintainable Software
Week 7 – 4:00pm Monday – 06 November 2023

valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

Topics for this Week

- Lecture 06A:
  - Version Control, Coding convention, Setting up Git with Intellij

- Lecture 06B:
  - More on GUI (JavaFX) – This is different from what was said in the lecture.

- Lab 06:
  - Setting up Git for your project

Module Feedback

Thank you for this. We really appreciated it

Early Module Feedback

- 21 % 92/435 – according to Moodle
- Positives
  - Lab
    - Good/fun and practical, love GUI
- Improvements
  - Content – more technical
  - Lab too long and repetitive
  - Repetition content on Java recap and UML
  - Lecture time

University of Nottingham
UK | CHINA | MALAYSIA

Version Control

Control your source code

Git and Repository Tools

- Git is a (free and open source) distributed version control system (from Linux)
- Designed originally for command line use
- Various GUI clients available
- And web front ends for management, such as GitLab

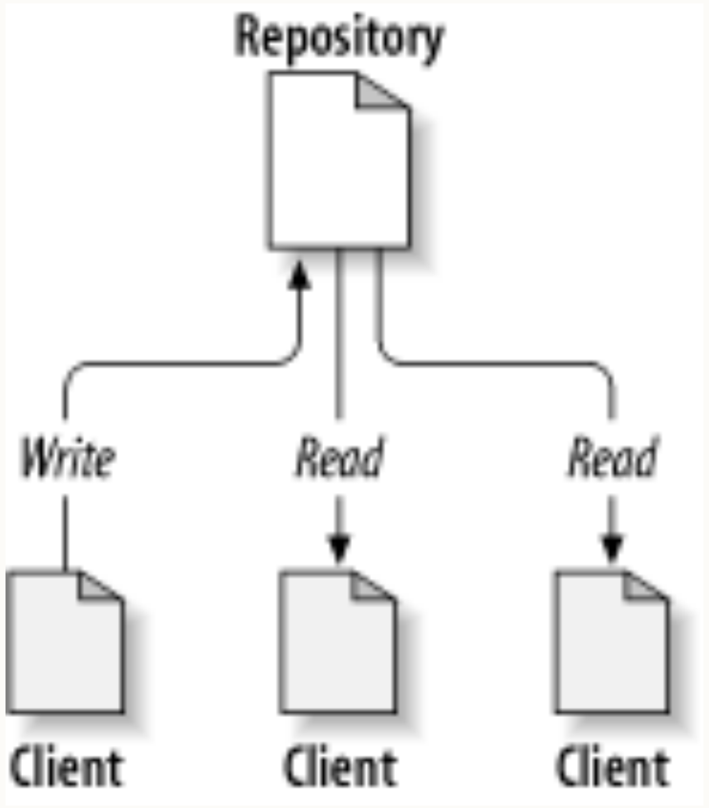
e,.g. http://projects.cs.nott.ac.uk/

Why Use Version Control?

- Track changes
  - Recover old versions
  - Examine source code history
- Works across networks (fosters collaboration)
- Similar with a networked file system + backup + additional functionality:
  - Tracks every change
  - Manages concurrency

Terminology: The Repository

- Stores a file system tree
- Remembers every change ever written to it

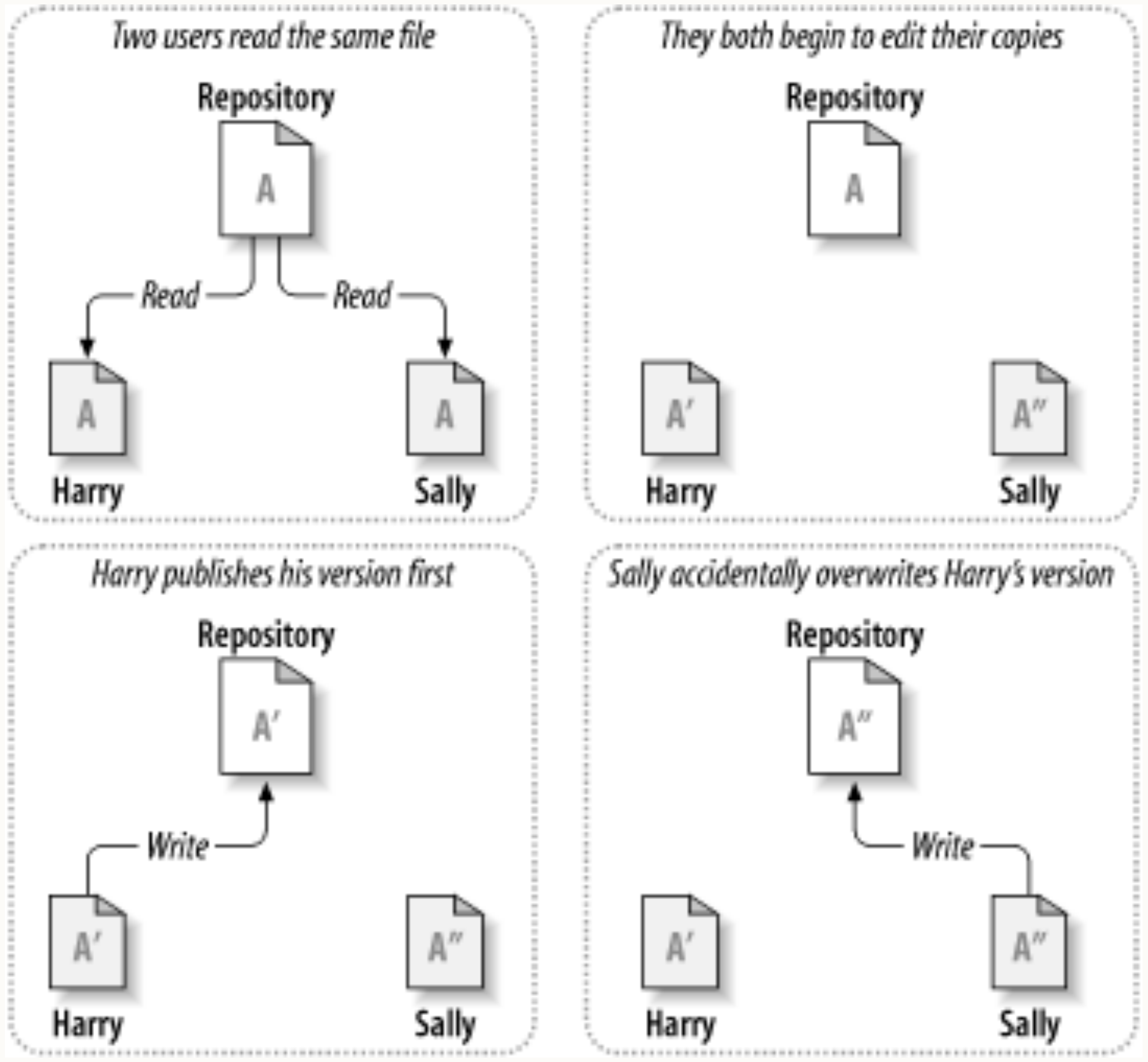


Concurrency Management

- Concurrency: simultaneous occurrence; coincidence - www.dictionary.com

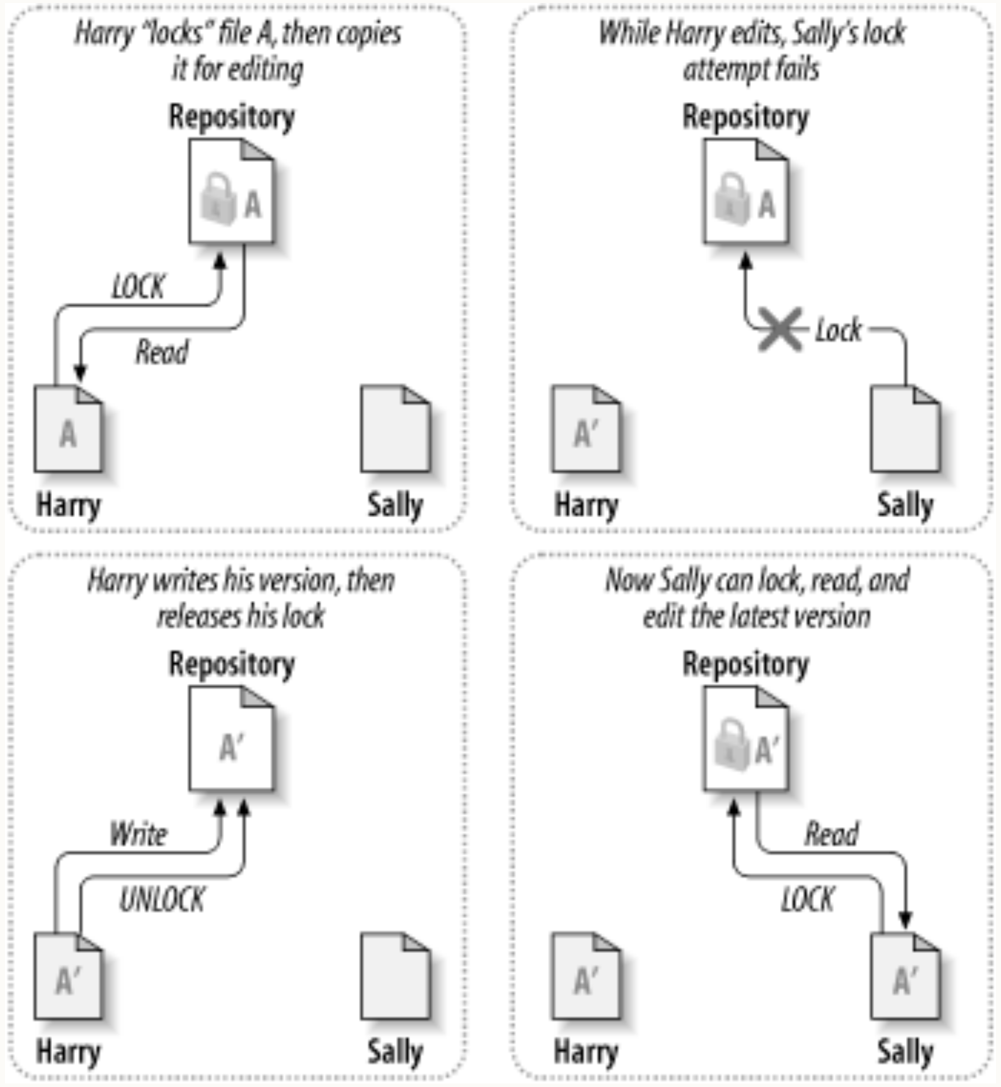
- Different ways to deal with concurrency
  - The problem of file sharing
  - Lock-modify-unlock solution
  - Copy-modify-merge solution

The Problem of File Sharing



Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

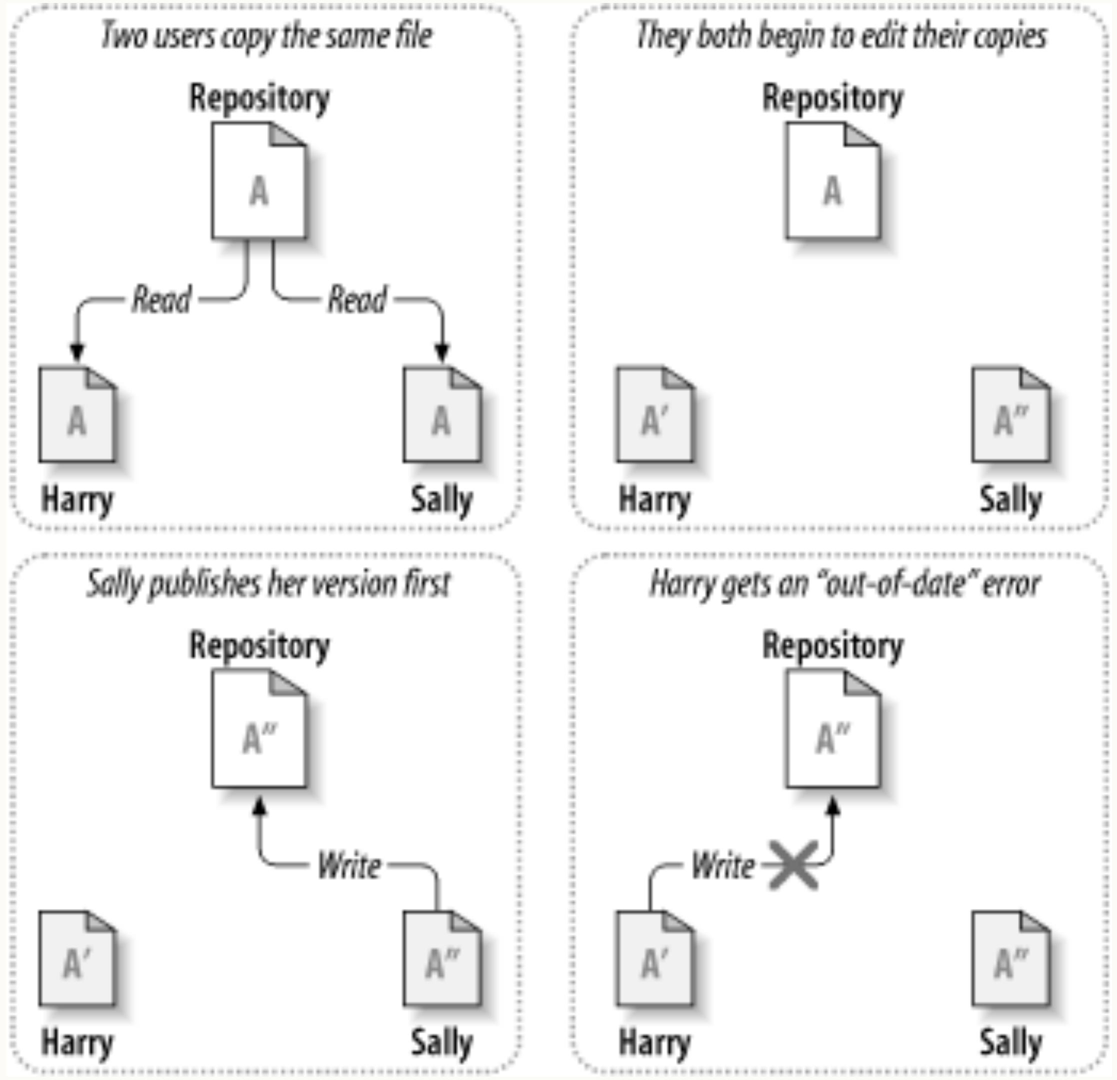
The Lock-Modify-Unlock Solution


Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
A
Harry
Sally
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
A′
Harry
Sally
Harry writes his version, then releases his lock
Repository
A′
Write
UNLOCK
A′
Harry
Sally
Now Sally can lock, read, and edit the latest version
Repository
A′
Read
LOCK
A′
A′
Harry
Sally

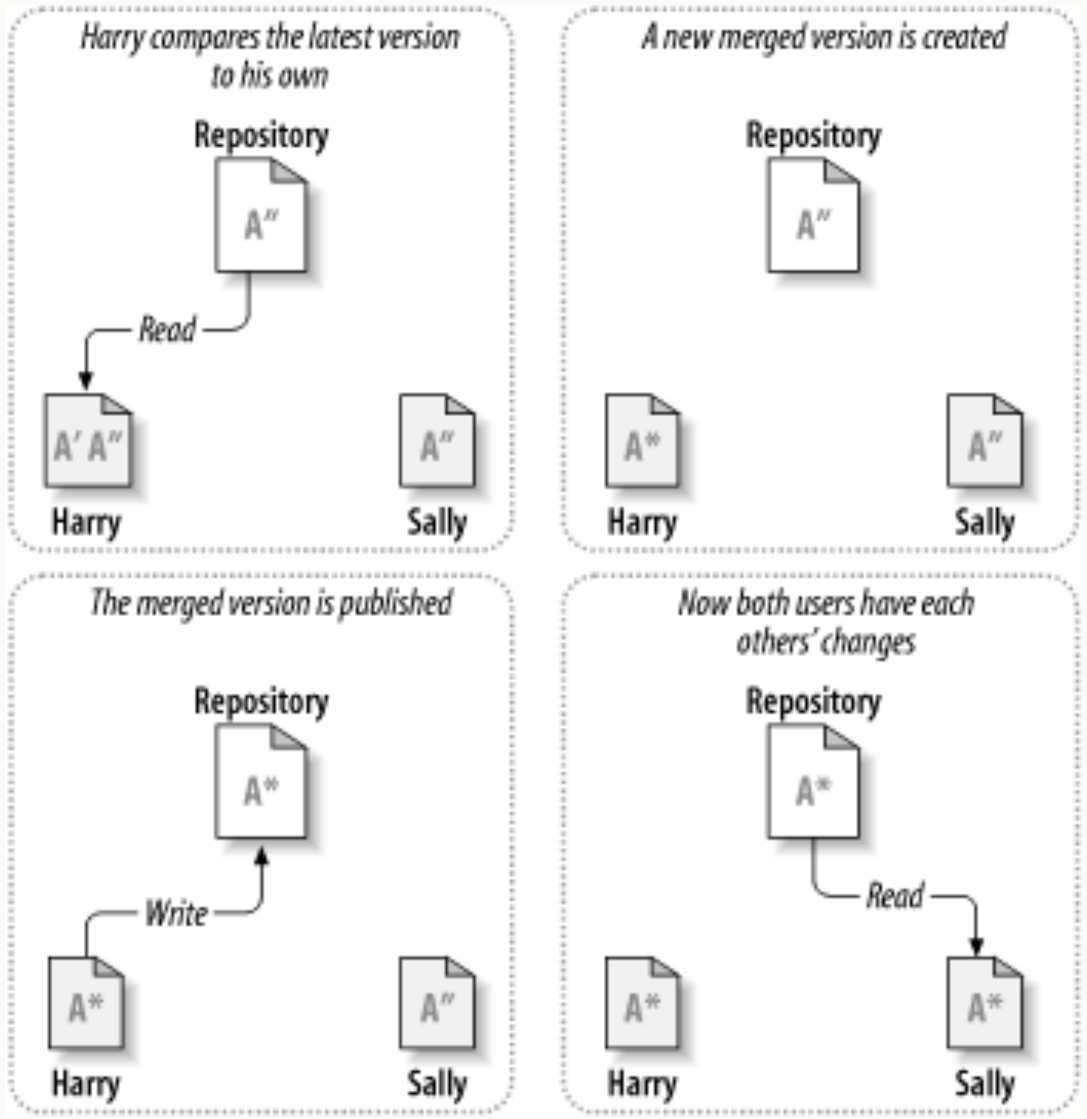
The Lock-Modify-Unlock Solution

- Problems
  - Harry locks a file and forgets about it. Then he goes on vacation.
  - Serialization
  - There is no protection for breaking dependencies between files. False sense of security.

The Copy-Modify-Merge Solution


Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Sally publishes her version first
Repository
A″
Write
A′
A″
Harry
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
A″
Harry
Sally

The Copy-Modify-Merge Solution


Harry compares the latest version to his own
Repository
A″
Read
A′ A″
Harry
A″
Sally
A new merged version is created
Repository
A″
A*
Harry
A″
Sally
The merged version is published
Repository
A*
Write
A*
Harry
A″
Sally
Now both users have each others' changes
Repository
A*
Read
A*
Harry
A*
Sally

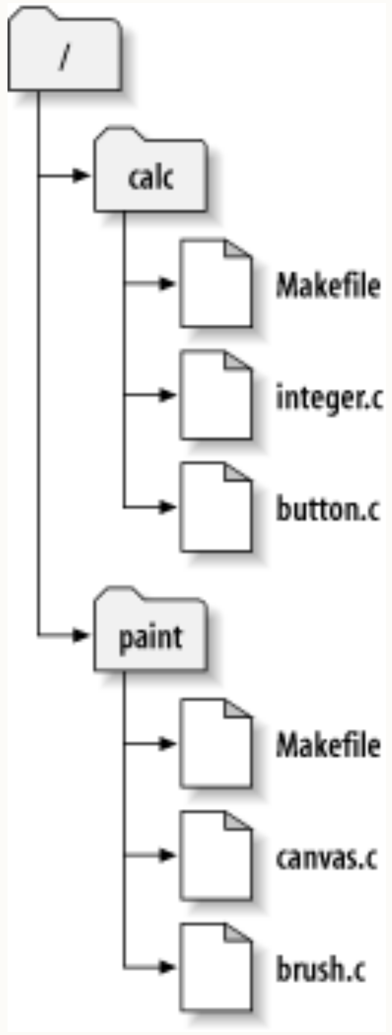
Concurrency Management

- The copy-modify-merge model: for text files
- Users work in parallel
- Concurrent changes are automatically merged.
- Conflicts are infrequent
- The lock-modify-unlock model: for binary files

Working Copies

- Regular directory tree
- It does not unless specifically told:
  - Incorporate other people's changes
  - Make your own changes available to others
- A typical repository = several projects.
- Each project = subdirectory
  - A working copy = one of those subdirectories.

Repository File System



Checkout

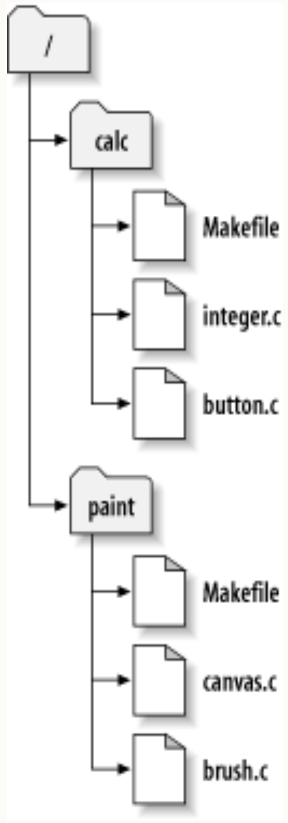
Create a private copy of project

```
$ git checkout http://svn.example.com/repos/calc
A    calc/Makefile
A    calc/integer.c
A    calc/button.c
Checked out revision 56.

$ ls -A calc
Makefile  button.c integer.c .svn/
```

Modify button.c

Repository file system



Commit

Publish your changes to others

```
$ git commit button.c -m "Fixed a typo in
    button.c."
Sending        button.c
Transmitting file data .
Committed revision 57.
```

Revisions

- A commit publishes changes to any number of files and directories.
- A commit is an atomic operation
- A commit = a new state of the repository's file system tree called a revision.
- Revision numbers: start with 0, increment for each commit.

Update

Incorporate changes that have been made since the last checkout (or update)

```
$ pwd
/home/sally/calc
$ ls -A
Makefile button.c integer.c .svn/
$ git push button.c
Updated to revision 57
```

Git

- Specify a project folder
- Build a new one; and then Git enables history of changes
- Git supports:
  - Creating repository
  - Committing code
  - File transfer back and forth
  - Clone or revert and manage history
- Works individually, but what if you need to work in a team?

Multi-user management with Git

- In the olden days “three way merge”
- You and your colleague want to create a commit which includes the changes of both parties
- If you have edited different parts of the file, GIT will know what to do, usually
- If you have edited the SAME part of the file, a good tool will show you both modifications and then you choose which one to go with
- Forking and branches….

Branching

- Each commit is a node in a linked list on disk
- Branch is the pointer to that node
- History tree is preserved by data structure
- Server have their own copies of branches
- CS server is configured to provide protection of master (can be turned off)

Valuable Git Resources For Homework

- ResourcesGit book available for free:
  Pro Git by Chacon and Straub
  https://git-scm.com/book/en/v2
- Tutorial:
  https://youtu.be/qvvq-NrForQ (step-by-step video demo)
  http://git-scm.com/docs/gittutorial
- Cheat sheets:
  http://jonas.nitro.dk/git/quick-reference.html
  http://rogerdudler.github.io/git- guide/files/git_cheat_sheet.pdf

University of Nottingham
UK | CHINA | MALAYSIA

Introduction to coding convention

- Bob’s coding convention
- Java Coding Conventions from Sun Microsystems

Introduction to coding conventions

- Writing a useful software application is difficult
- First implementing larger, long-term project.
- Maintainable software requires more effort than creating new software
- Be systematic and follow good practice

Why Coding Conventions?

- Illegible code is default-quickly turns into *legacy* code
- In “reality” most software projects fail [Ellis 2008, Krigsman 2008] Basic philosophy behind conventions is to maximize legibility
- Legible software is better software
- Legible software contains fewer bugs, more stable Legible software is more flexible, encourages re-use
- Two other key ingredients:
  - Software Design
  - Comment Conventions

Bob’s Rule 1: Method Length (75 lines or less)

- Method is visible on a single screen/page.
- Possible to see whole method from start to finish (without scrolling).
- Except: Methods with switch statements and perhaps main method.
- The less re-usable and more difficult it is to modify.

Bob's Rule 1: Method Length (75 lines or less)

- More likely it is to contain bugs and more difficult it is to debug.
- By confining method to one screen, it gives programmer (at least)a chance to keep track of variables from beginning to end.
- Conformance to this rule facilitates code optimization with profiler [Meyers '96]

Rule 2: Indentation

- No methods shall use more than five levels of indentation.
- Too many levels of indentation quickly renders code illegible.

Rule 3: Line Length below 80 characters

- It should not be necessary to expand code editor to entire screen width in order to read single line of code.

- Lines that are too long are less legible and more difficult to debug.

- The longer a line is, the more difficult it is for eyes to move from end of one line to next.

- Good publishers use a guideline of approximately 66 characters per line of text (so 80 is generally too much) [Oetiker et al, 2008].

Rule 3: Line Length below 80 characters

- It should not be necessary to expand code editor to entire screen width in order to read single line of code.
- Lines that are too long are less legible and more difficult to debug.
- The longer a line is, the more difficult it is for eyes to move from end of one line to next.
- Good publishers use a guideline of approximately 66 characters per line of text (so 80 is generally too much) [Oetiker et al, 2008].

Rule 4: Class Variable Names

- Class variables should be easily distinguishable from local variables or other types of variables.
- All class variables start with the two character sequence "m_"(as in "member" variable) e.g., m_ClassVariable.
- Except: symbolic constants. Symbolic constants are written in ALL_CAPITALS.

Rule 5: Accessor Methods

- Enforces encapsulation: extremely important concept in object- oriented methodology. (Wirfs-Brock et al. '90)
- Accessing member variables with methods makes implementation easy to change, e.g., a float to an int.
- Prevents unwieldy (or even impossible) search-and-replace operations [VTK Coding Standards '09, Sun Microsystems '99].
- All class variables are accessed with accessor methods, i.e. Get() and Set() methods, e.g., GetClassVariable(), SetClassVariable(int newValue)

Rule 6: Accessor Methods

- Accessor methods are most common to use, as such, it is most convenient when defined at the "top" of the file or class definition.

- Accessor methods come at top of both header files and implementation files.

Rule 7: Class Variables

- All member class variables are private.
- Keeping class variables private enforces encapsulation.
- Only the class itself should know about the specific implementation details of its own data [Meyers 2005]
- Except: symbolic constants

Rule 8: Method Naming

- It is very nice to tell whether method is private or public simply by looking at it (without having to look it up) [Sun Microsystems 1999]. Even in presence of tools.
- Private methods begin with a lower-case letter.
- Public methods begin with an upper-case letter.

Rule 9: Method Parameters

- Do not require more than 5 parameters. Too may suggests problem with software design
- The more parameters a method takes, the less re-usable it is.
- Have different implementations of same method taking different (but only a few)
- A long list of parameters may indicate that changes to design are necessary, e.g., the introduction of a new class(es) or re- arrangement of existing classes [Sun Microsystems 1999].

Rule 10: Symbolic Constants

- Do not use numbers in your code, but rather symbolic constants.
- One 6 may not be same as another 6. [Sutter and Alexandrescu 2005]
- Using symbolic constants instead of typing numbers makes code much more legible.
- Even original author eventually forgets what number is. Values of symbolic constants are easy to change.
- Changing values of numbers directly in the code causes bugs, especially when the number appears in multiple places [Sun Microsystems 1999].
- Horstmann articulates rule as “Do Not Use Magic Numbers” [Horstmann 2003].

Rule 10: Symbolic Constants Example with Magic Numbers

```
void RSL_OglTexture::CopyImageData(FXuchar* textureData) {

  bool debug = false;
  int currentRow, currentCol, textureOffset, dataOffset; int lengthOfOneRow
                          = this->GetWidth();
  int heightOfOneColumn = this->GetHeight();

  if (debug) {
    cerr << "RSL_OglTexture::CopyImageData() name: " << this->GetName() << endl; cerr << " width: " << this-
    >GetWidth() << ", height: " << this->GetHeight() << endl;
  }
  for (currentRow = 0; currentRow < heightOfOneColumn; currentRow++) { for (currentCol = 0;
    currentCol < lengthOfOneRow; currentCol++) {

      textureOffset = currentRow * lengthOfOneRow * 4 + currentCol * 4; dataOffset = currentRow

      * lengthOfOneRow * 3 + currentCol *3;

      this->GetBufferDataPtr()[textureOffset    +    0]    =    textureData[dataOffset    +    0];        this-
      >GetBufferDataPtr()[textureOffset     +     1]     =     textureData[dataOffset     +     1];          this-
      >GetBufferDataPtr()[textureOffset     +     2]     =     textureData[dataOffset     +     2];          this-
      >GetBufferDataPtr()[textureOffset + 3] = 255;
    }
  }
  if (debug) cerr << "RSL_OglTexture::CopyImageData() END" << endl;
```

Rule10: Example

```
void RSL_OglTexture::CopyImageData(FXuchar* textureData) {

  bool debug = false;
  int currentRow, currentCol, textureOffset, dataOffset; int
  lengthOfOneRow           = this->GetWidth();
  int heightOfOneColumn = this->GetHeight();

  if (debug) {
    cerr << "RSL_OglTexture::CopyImageData() name: " << this->GetName() << endl; cerr
    << " width: " << this->GetWidth() << ", height: " << this->GetHeight() << endl;
  }
  for (currentRow = 0; currentRow < heightOfOneColumn; currentRow++) { for
    (currentCol = 0; currentCol < lengthOfOneRow; currentCol++) {

      textureOffset = currentRow * lengthOfOneRow * NUM_RGBA_COMPONENTS +
                   currentCol * NUM_RGBA_COMPONENTS;
      dataOffset = currentRow * lengthOfOneRow * NUM_RGB_COMPONENTS +
                 currentCol * NUM_RGB_COMPONENTS;

      this->GetBufferDataPtr()[textureOffset + 0] = textureData[dataOffset + 0];  this-
      >GetBufferDataPtr()[textureOffset  +  1]  =  textureData[dataOffset  +  1];   this-
      >GetBufferDataPtr()[textureOffset  +  2]  =  textureData[dataOffset  +  2];   this-
      >GetBufferDataPtr()[textureOffset + 3] = MAX_ALPHA;
    }
  }
  if (debug) cerr << "RSL_OglTexture::CopyImageData() END" << endl;
```

Javadoc Documentation

Going beyond manual code comments

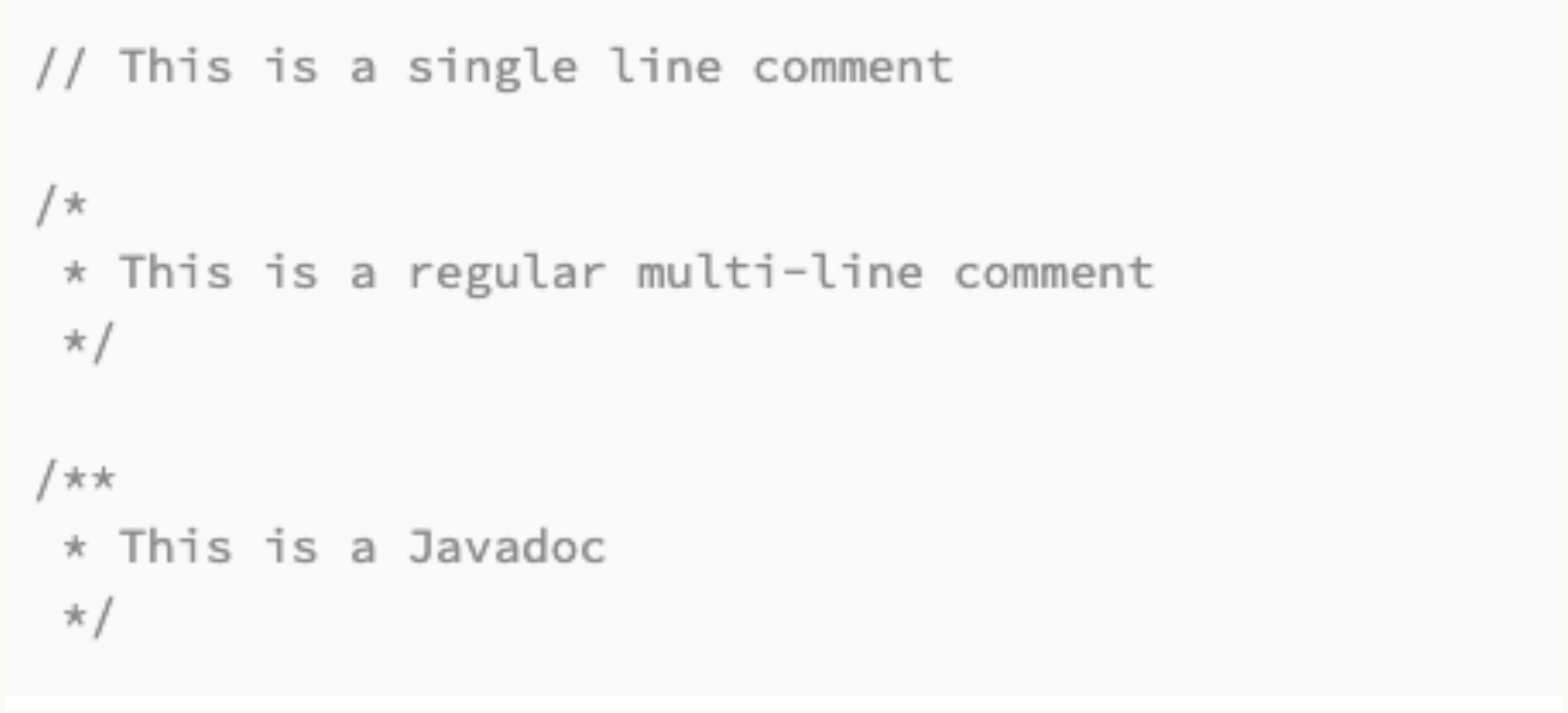
- Code comments are essential for maintenance as they are key to having another person be able to understand what you have done
- Semi-Automatic documentation enables:
  - Standard comment formatting and structure
  - Less typing, some automation
- Examples include Doxygen and Javadoc
  - Doxygen can be used for C++, with modified versions for C#
  - Can also be used in conjunction with the python live editor
  - Javadoc for Java

IDEs are really helpful for comments

- Many tools are built into IDEs
- It is great for helping us create maintainable code including in-built testing help
- Javadoc is a great tool for java documentation
  - This comes with the JDK and requires you to tag your code with special comments

What is Javadoc

- Similar to multi-line comment



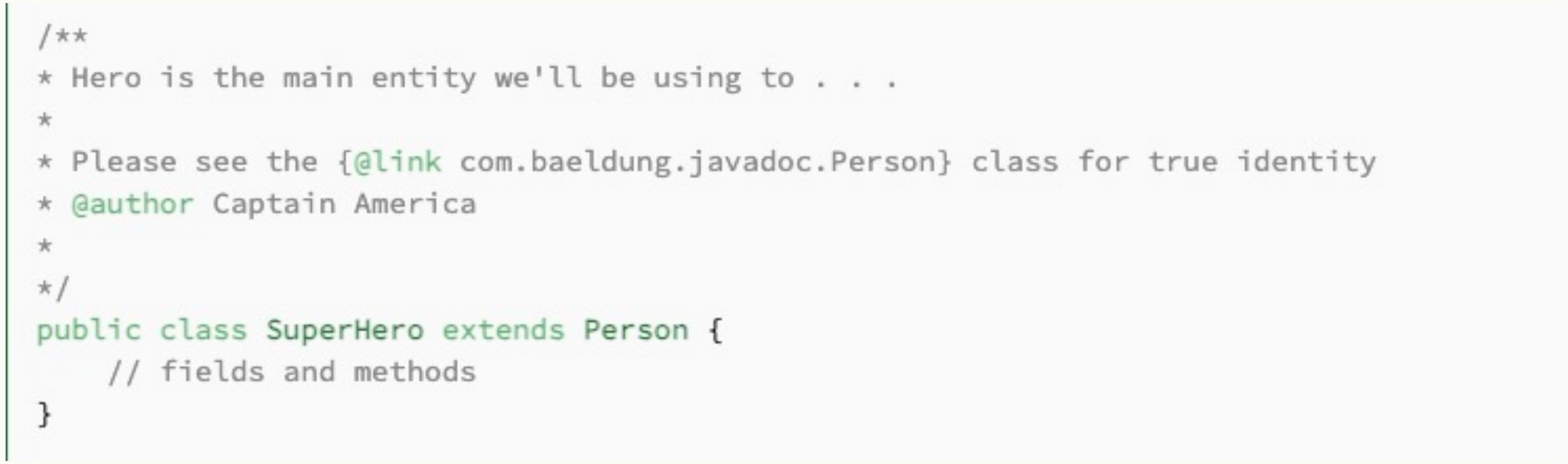
```
// This is a single line comment

/*
 * This is a regular multi-line comment
 */

/**
 * This is a Javadoc
 */
```

What is Javadoc?

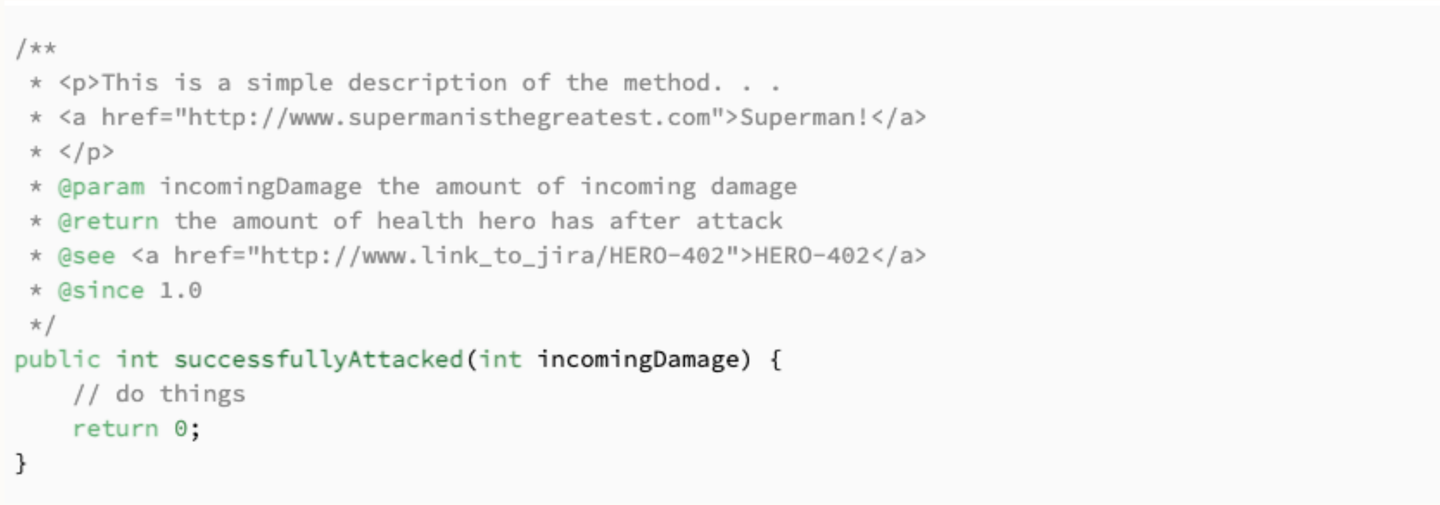
- Describe what you are commenting about
- Standard block tag marked with ”@” symbol, which describes specific meta-data
- **Class level** – see the inline comments @link and @author



```
/**
* Hero is the main entity we'll be using to . . .
*
* Please see the {@link com.baeldung.javadoc.Person} class for true identity
* @author Captain America
*
*/
public class SuperHero extends Person {
    // fields and methods
}
```

What is Javadoc? – Method level

- *@param* provides any useful description about a method's parameter or input it should expect
- *@return* provides a description of what a method will or can return
- *@see* will generate a link similar to the *{@link}* tag, but more in the context of a reference and not inline
- *@since* specifies which version of the class, field, or method was added to the project
- *@version* specifies the version of the software, commonly used with %I% and %G% macros
- *@throws* is used to further explain the cases the software would expect an exception
- *@deprecated* gives an explanation of why code was deprecated, when it may have been deprecated, and what the alternatives are



```
/**
 * <p>This is a simple description of the method. . .
 * <a href="http://www.supermanisthegreatest.com">Superman!</a>
 * </p>
 * @param incomingDamage the amount of incoming damage
 * @return the amount of health hero has after attack
 * @see <a href="http://www.link_to_jira/HERO-402">HERO-402</a>
 * @since 1.0
 */
public int successfullyAttacked(int incomingDamage) {
    // do things
    return 0;
}
```

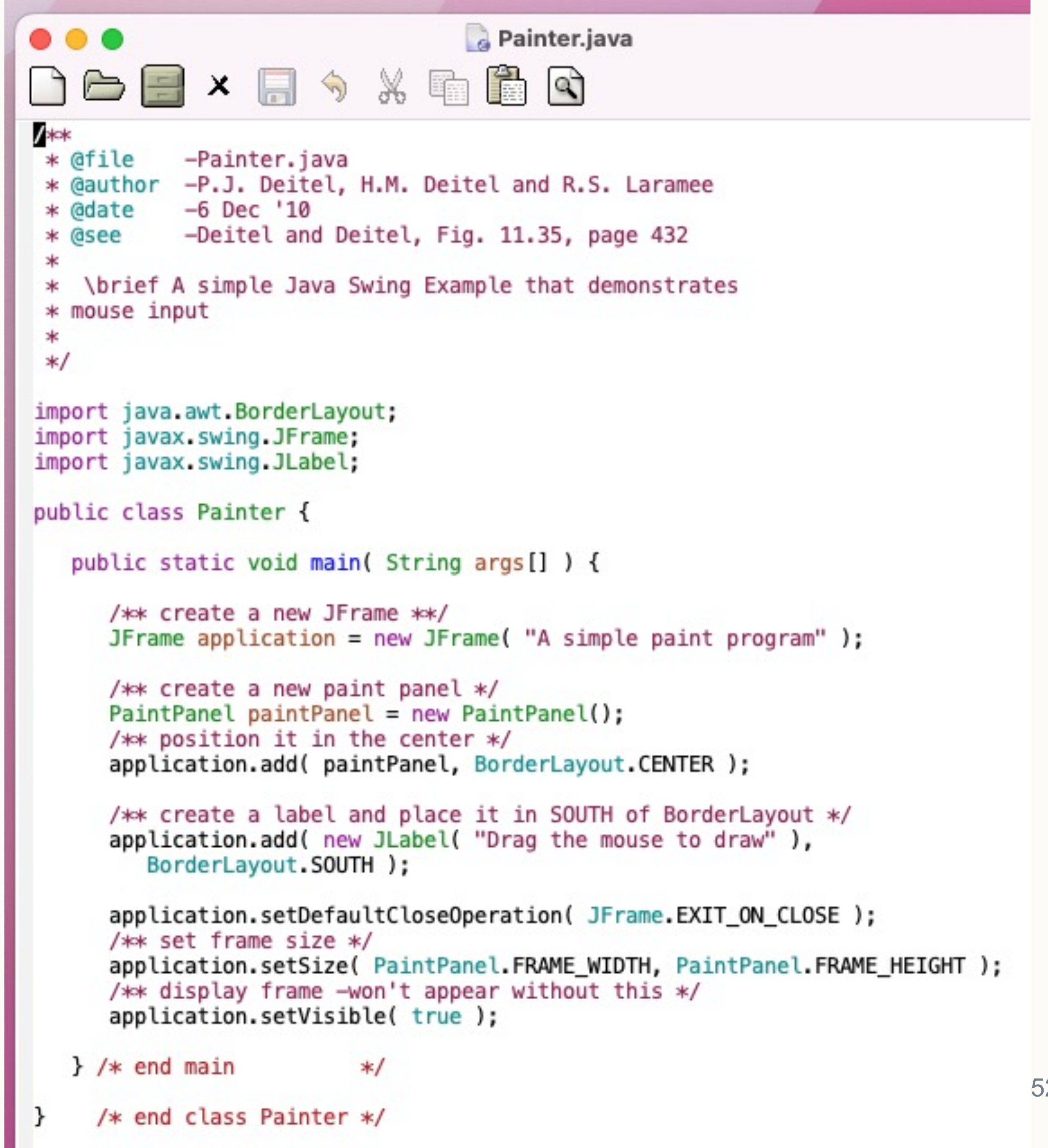
https://www.baeldung.com/javadoc

Useful Javadoc Tags

- Syntax: @<tag>
- It generates a really easy to use HTML based output as a living document
- Updated each time you compile if Javadoc is in the compilation path
- Some useful tags:
  - @param: to explain a method parameter
  - @return: to annotate a method return value
  - @throws/@exception: for your exception handling
  - @deprecated: bits of the code you no longer use
  - {@code}: puts syntax in your documentation

Javadoc Example

https://www.oracle.com/uk/technical-resources/articles/java/javadoc-tool.html



```
/**
 * @file    -Painter.java
 * @author  -P.J. Deitel, H.M. Deitel and R.S. Laramee
 * @date    -6 Dec '10
 * @see     -Deitel and Deitel, Fig. 11.35, page 432
 *
 *  \brief A simple Java Swing Example that demonstrates
 * mouse input
 *
 */

import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Painter {

   public static void main( String args[] ) {

      /** create a new JFrame **/
      JFrame application = new JFrame( "A simple paint program" );

      /** create a new paint panel */
      PaintPanel paintPanel = new PaintPanel();
      /** position it in the center */
      application.add( paintPanel, BorderLayout.CENTER );

      /** create a label and place it in SOUTH of BorderLayout */
      application.add( new JLabel( "Drag the mouse to draw" ),
         BorderLayout.SOUTH );

      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      /** set frame size */
      application.setSize( PaintPanel.FRAME_WIDTH, PaintPanel.FRAME_HEIGHT );
      /** display frame -won't appear without this */
      application.setVisible( true );

   } /* end main           */

}    /* end class Painter */
```

Acknowledgements

We thank Julie Greensmith and Dan Lipsa for lecture material.

More information can be found in:

- Hans van Vliet**, Software Engineering: Principles and Practice,** 3rd Edition, 2008, John Wiley & Sons
- FreeTechBooks.com

References

- Bob's Concise Coding Conventions, Robert S. Laramee
- The Visualization Toolkit (VTK) Coding Conventions
- Java Coding Conventions from Sun Microsystems

- S. Meyers. **More Effective C++, 35 New Ways to Improve Your Programs and Design**, Addison-Wesley, 1996 (336 pages)
- S. Meyers. **Effective C++, 55 Specific Ways to Improve Your Programs and Designs**, Addison-Wesley, 2005 (320 pages)
- H. Sutter and A. Alexandrescu, **C++ Coding Standards, 101 Rules, Guidelines, and Best Practices**, Addison-Wesley, 2005 (220 pages)
- B. Stroustrup, **The C++ Programming Language, Special Edition**, Addison-Wesley, 2000 (1018 pages)