



Introduction –Developing Maintainable Software

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior



Overview

- Introduction to the team for COMP2013
- Learning and Teaching Structure
- Explore your previous knowledge
- About this module
- This week!
- Software Quality - Maintainable Software
- Software Maintenance (high level)
- What is actually done in software maintenance?
- Final thoughts...



Dr Marjahan Begum – Assistant Professor

- PhD Computer Science Education from University of Nottingham (back in the days)
- Learning scientist interested in Computer Science
- Psychology of Computer Science
- Women in Computing
- Diversity in Computing
- Human aspect of software engineering (software teams)





Dr Horia A. Maior - Assistant Professor

- PhD and Postdoctoral from University of Nottingham
- Around since 2012
- Taught in Ningbo Campus in China

My research intersect:

- Human Computer Interaction
- Brain Computer Interfaces
- Human Robot Interaction
- Human AI Interaction
- Responsible Research and Innovation

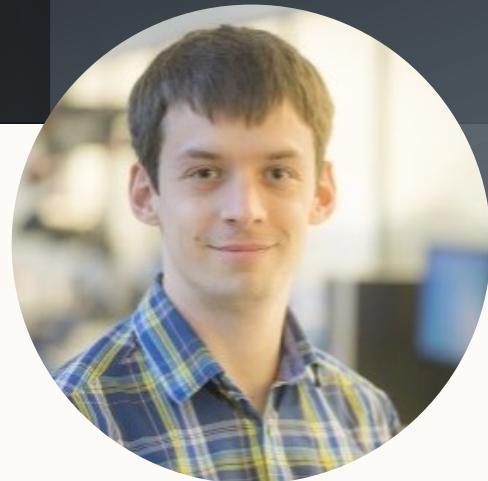
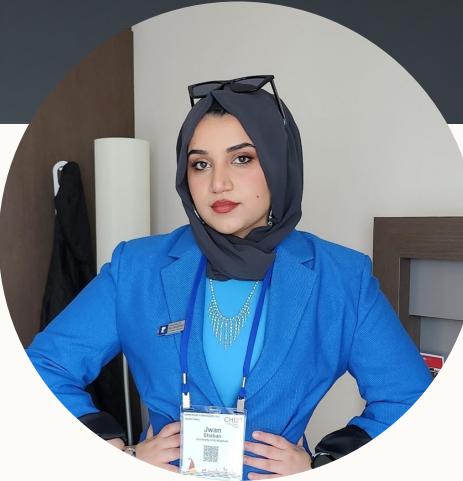




Amazing team of Teaching Support

- Dalma Kilic (PhD) – Teaching Associate
- Jwan Shaban – PhD
- Lewis Stuart – PhD
- Warren Jackson (PhD) – Teaching Associate
- Weiyao Meng(PhD) – Teaching Associate

Teaching Associate and PhDs



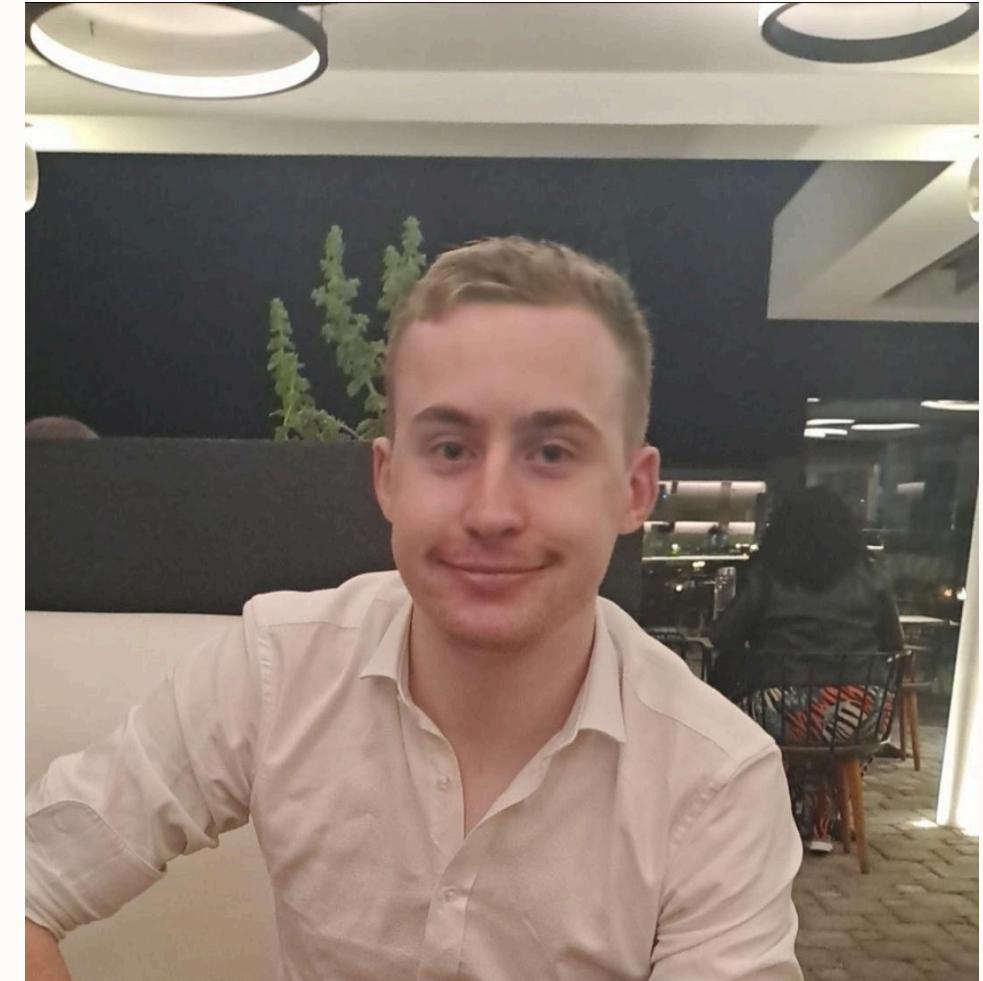
- Weiyao Meng – Teaching Associate
- Jwan Shaban – PhD
- Warren Jackson – Teaching Associate
- Dalma Kilic – Teaching Associate
- Lewis Stuart – PhD
- We will introduce all in the lab

Dalma Kilic



Lewis Stuart

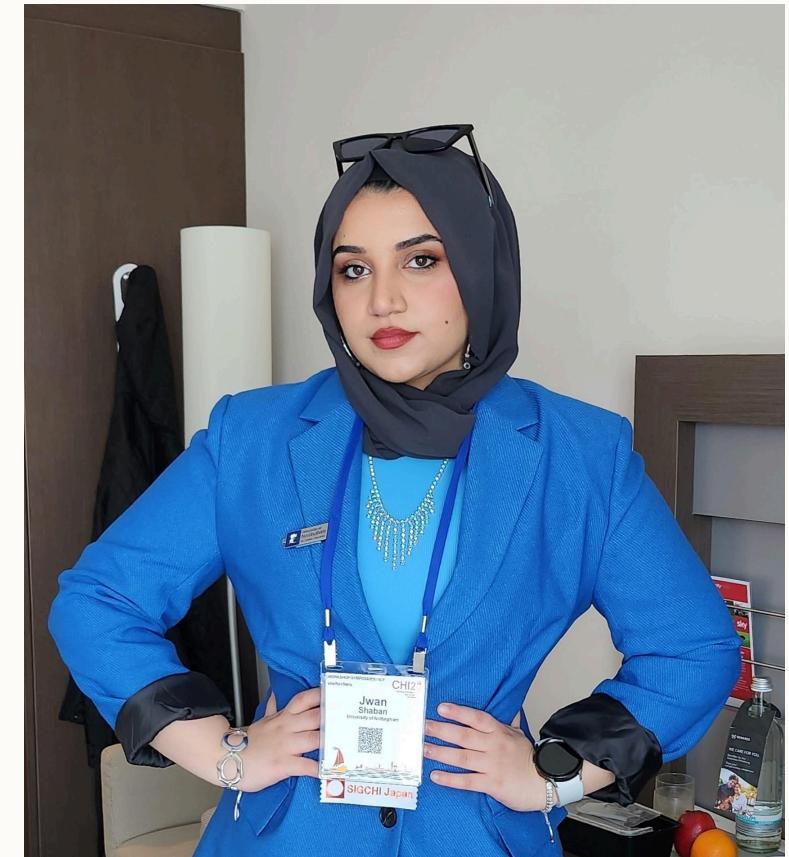
- Leeds University (first MEng Advanced Computer Science).
- Computer Vision Lab 3D reconstruction of plants from multiple views, using a UR5 robotic arm to capture data automatically.
- Taught on the Computer Graphics (COMP3011) module.





Jwan Shaban

- Tracking Mental Workload and personal informatics.
- BSc in software engineering from Manchester met University and a Masters in computer science from Nottingham.





Dr. Weiyao Meng

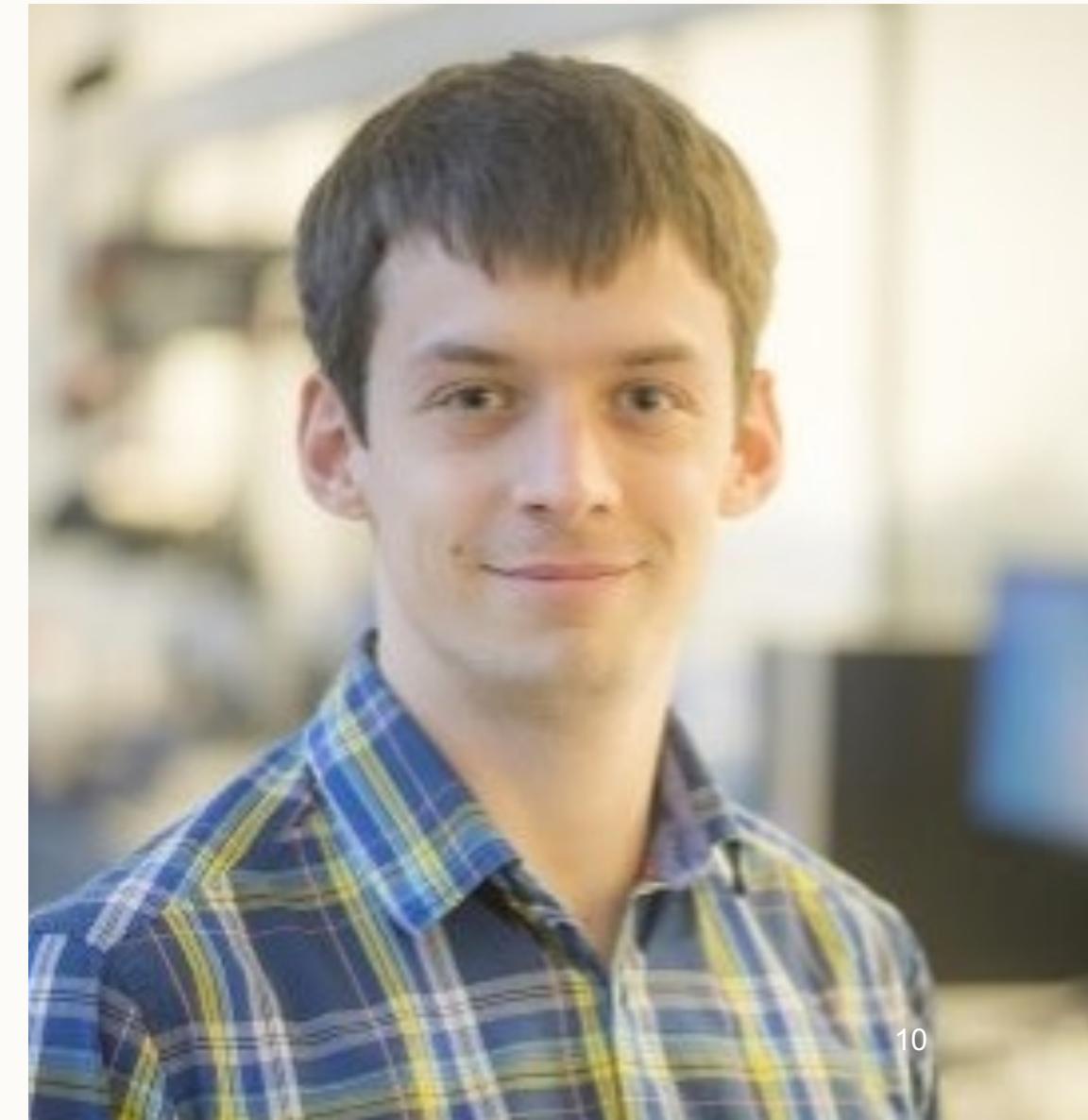
- PhD at the Computational Optimisation and Learning (COL) Lab, machine learning for Vehicle Routing Problems with Time Windows.
- Co-convened COMP1008 (Fundamentals of Artificial Intelligence) in 2022-23 and has previously supported the Mathematics for Computer Scientists, Data Modelling and Analysis, Simulation and Optimisation for Decision Support.





Warren Jackson – Teaching Associates

- Experienced teacher in Computer Science
- PhD student within the Computational Optimisation and Learning (COL)





Learning and Teaching Structure



Office Hours

Dr Marjahan Begum

Tuesdays: 14:00 to
15:00

Room: B72

Dr Horia Maior

Thursdays: 14:00 to
15:00

Geospatial Engineering
Building (3rd floor)



Learning and Teaching Structure

- **Lecture A** is on Mondays : the Jubilee Campus Business School South Auditorium from 16:00 to 18:00.
- **Lecture B** is on Thursdays 5th: the Jubilee Campus Business School South Auditorium from 13:00 to 14:00
- **Lab** is Fridays (A07, A32, Computer Science Atrium).



Assessment

- Coursework (75%)
 - Individual coursework
 - Refactor and extend an existing code base
 - We aim to release it by the end of week 5
 - A milestone setting up infrastructure and design documentations week 8
- Exam (25%)
 - ROGO exam (90 minutes)



How to participate

- All communication will be through Moodle forum and teams
- Some individuals email will be answered depending on circumstances
- Team channel – please help each other without compromising academic integrity



ChatGPT



Tell me about your
previous knowledge and
skills?



Polls

- Tell me technologies you are familiar with from year 1 and previous experience?
- How do you think this module is related to previous years?



Do these people remind you of something??

- You have had Java overview lectures from Jason
- Max has introduced you to Agile
- Graham has enticed you with his Haskell
- Some of you are very accomplished programmers
- You will have seen some of the content before



- Practical and prepares you for the real job..





Previous topics you will have covered

- Programming in Java:
 - Classes and interfaces
 - Objects
 - Arrays
 - Strings
 - Inheritance
 - Java Collections
 - Dealing with exceptions
 - Software testing
 - Refactoring
- Tools





Previous topics you will have covered

- Software Engineering
 - Requirements Engineering
 - Prototyping
 - Object Oriented Design
 - Implementation Strategies
 - Debugging
 - Release, Acceptance, and User Testing
 - Software Quality
 - Software Evolution
 - Agile Methodologies
 - Project Management



So ... What is the difference?



**What do you want to get
out of this module?**



The overall learning objectives of this module

- To ensure you understand that software maintenance is central to software development
 - Maintainable software and software maintenance is essential to developing large codebases and code written and updated by multiple people
 - Software can only be maintained if good practices of coding and design are followed
- To ensure you can write maintainable code
 - So that you understand that software maintenance is not an afterthought in the software process models and lifecycle
- To transform you from a 'coding caterpillar' to a 'developer butterfly'

This quote is from Julie :)



COMP2013 is all about BIG software projects

- It's about ...
 - How to create them from scratch so you can maintain them more easily later
 - How to approach an existing large mass/mess of code!
 - The tools you need to maintain this code "collaboratively"
- All this is very practical advice that you will find useful in the real world
 - We (hopefully) will have some guest lecturers to provide a real-world take on this
 - It should give you lots of pointers for COMP2002 (the group project)



How best to get most out of this module?

- Ask questions and clarify the requirements for this module through Team's channel and during the lab.
- Engage with the in lectures and lab by asking questions, attempt the lab and come to labs with questions, read the coursework line by line and clarify where required.
- Please keep in mind: The module is not about teaching you the latest Java and software engineering technologies and libraries, but **it's about learning the principles** of "how to develop maintainable software" and "how to maintain software".



What tools, languages and software?



Relevant Software

- Software we use in this module:
 - Java 12+ (requires 64 bit system > use the virtual desktop if you don't have a 64 bit system)
 - IntelliJ IDEA (this is the IDE to be used for your coursework)
 - Visual Paradigm (also available as online solution, but requires registration)
 - JavaFX 12+ graphics library and Gluon Scene Builder
 - Maven/Gradle
 - Git
- Our recommendation: Use the latest (Long-Term-Support) versions



What are we doing this week?



Topics for this Week

- Lecture 01A
 - Welcome to the module
 - Developing maintainable software
 - Why COMP2013?
 - Challenges of software maintenance
- Lecture 01B
 - OO and Java programming refresher
 - Lab sheet
- Lab 01
 - IntelliJ basics
 - Practicing Java basics
 - Working with existing cod



Learning Outcomes for this Week

- At the end of this week ...
- You should understand what developing maintainable software is about
- You should understand why this module is useful
- You should be set up for future lab sessions
- You should have a good working knowledge of the basics of the object-oriented concepts, Java, and the IntelliJ IDE



What do you think
developing maintainable
software is about?



What are the relationships?

**Maintainable
Software (Qualities)**

**Software
Maintenance**



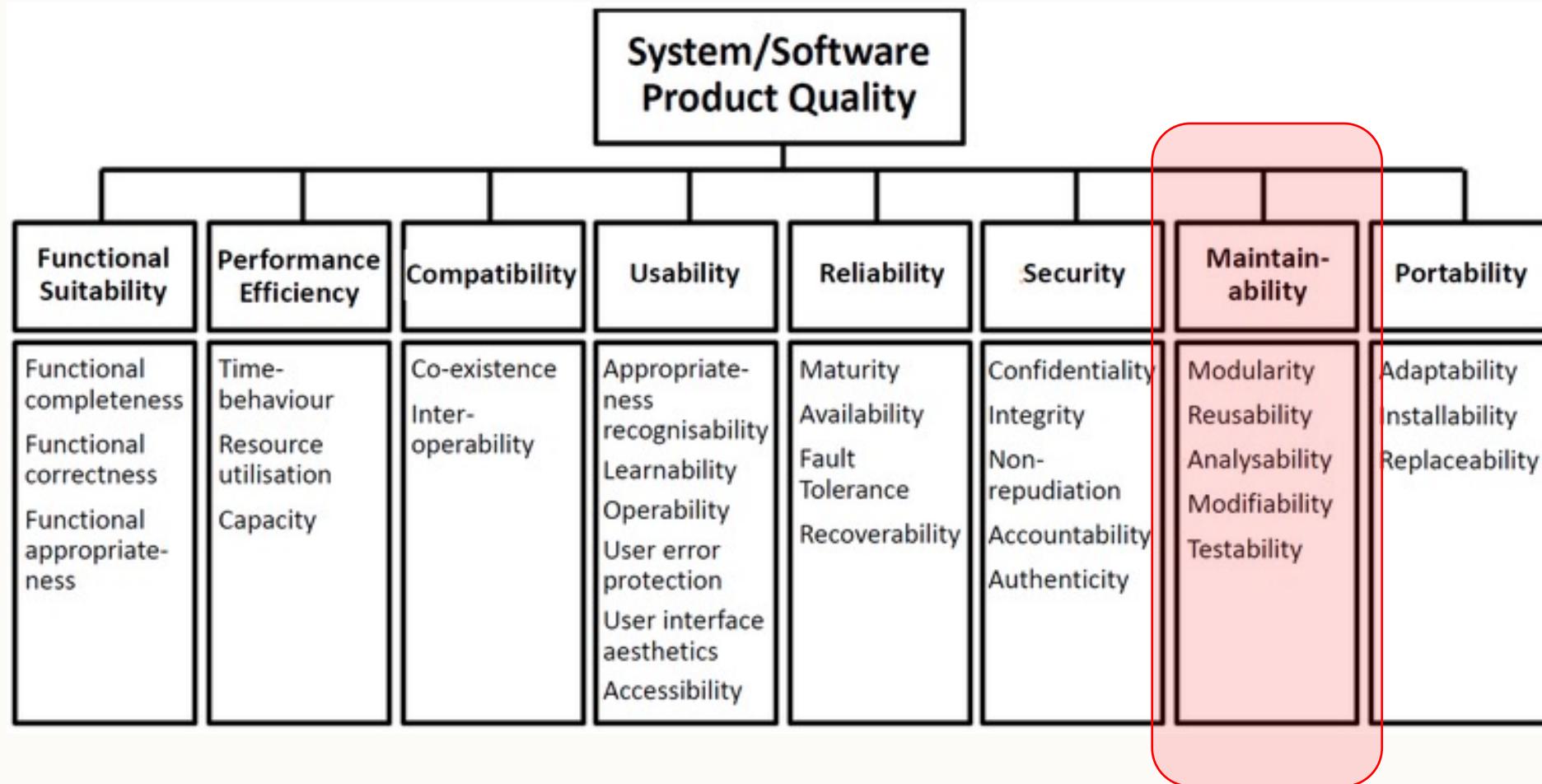
Maintainable software

Software Quality – principles and guidelines – wish list



Developing Maintainable Software

■ ISO 25010





Developing Sustainable Software

Home / ISO 25000 STANDARDS / ISO 25010

EN ES

Maintainability

This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements. This characteristic is composed of the following sub-characteristics:

- **Modularity** - Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
- **Reusability** - Degree to which an asset can be used in more than one system, or in building other assets.
- **Analysability** - Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
- **Modifiability** - Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
- **Testability** - Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.

<https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/57-maintainability>



Maintainability

- **Modularity**
 - Discrete components
 - Change to one component have minimal impact on others
 - Low ripple effect
- **Reusability**
 - Reuse Reuse Reuse
- **Analysability**
 - Effectively and Efficiently assess impact of individual component



Maintainability

- **Modifiability**
 - Effectively and efficiently modify code
 - Without defects/degrading product quality
- **Testability**
 - Effectively and efficiently write test criteria
 - Test can be performed to evaluate software qualities



Developing Maintainable Software

- Three principles for developing maintainable software:
 - Maintainability benefits most from adhering to simple guidelines
 - e.g. never write methods that have more than 15 lines of code
 - Maintainability is not an afterthought, but should be addressed from the very beginning of a development project
 - Some violations are worse than others; the more a software system complies with the guidelines, the more maintainable it is.

Visscher et al (2016)



Developing Maintainable Software

- Some simple guidelines to develop maintainable software:
 1. Write short units (constructors/methods) of code
 2. Write simple units of code
 3. Write code once
 4. Keep unit interfaces small
 5. Separate concerns in modules (classes)
 6. Couple architecture components loosely
 7. Keep your codebase small
 8. Automate your development pipeline and tests
 9. Write clean code



Visser et al (2016)



A Poem

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.^[a]
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.^[b]
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!



Software Maintenance

Software Quality – checking of it the maintainability



Software Maintenance

- What does it involve?
 - "Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."
- But we also need to consider how we can reduce the effort of maintenance!
 - Building software that is easy to maintain and extend in the first place!

IEEE Standard for Software Maintenance:
<https://ieeexplore.ieee.org/document/257623>



Resources Spend on Initial Development vs. Maintenance

- It's important to learn about maintenance
- It's important to build maintainable software

Author	Resources spend	
	initial development	maintenance
Daniel D. Galorath	25%	75%
Stephen R. Schach	33%	67%
Thomas M. Pigoski	<20%	>80%
Robert L. Glass	20-60%	40% - 80%
Jussi Koskinen	<10%	>90%

<http://blog.lookfar.com/blog/2016/10/21/software-maintenance-understanding-and-estimating-costs/>



Resources Spend on Initial Development vs. Maintenance

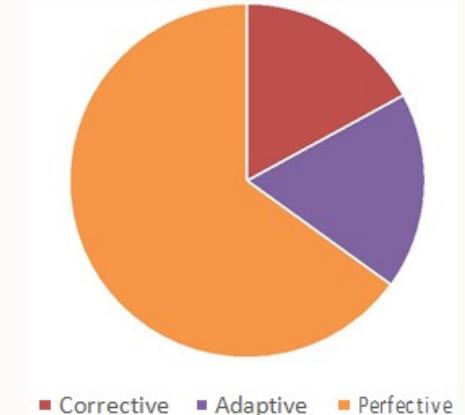




Three (or four, depending on authors) Main Categories of Maintenance

- Corrective Maintenance
 - Finding and fixing errors in the system
 - Example: Fixing bugs
- Adaptive Maintenance
 - The system has to be adapted to changes in the environment in which it operates
 - Example: Updating links to resources after change
- Perfective + Preventive Maintenance
 - Users of the system (and/or other stakeholders) have new or changed requirements
 - Ways are identified to increase quality and prevent future bugs from occurring

Maintenance effort



■ Corrective ■ Adaptive ■ Perfective



Actually there can be lots more!

- Maintenance vs Evolution?
- Maintenance:
 - Preserving software in a working state
- Evolution:
 - Improving software (e.g changing requirement)
- What we will learn will largely be applicable to both

30 TAXONOMY OF SOFTWARE MAINTENANCE AND EVOLUTION

TABLE 2.1 Evidence-Based 12 Mutually Exclusive Maintenance Types

Types of Maintenance	Definitions
Training	This means training the stakeholders about the implementation of the system.
Consultive	In this type, cost and length of time are estimated for maintenance work, personnel run a help desk, customers are assisted to prepare maintenance work requests, and personnel make expert knowledge about the available resources and the system to others in the organization to improve efficiency.
Evaluative	In this type, common activities include reviewing the program code and documentations, examining the ripple effect of a proposed change, designing and executing tests, examining the programming support provided by the operating system, and finding the required data and debugging.
Reformative	Ordinary activities in this type improve the readability of the documentation, make the documentation consistent with other changes in the system, prepare training materials, and add entries to a data dictionary.
Updative	Ordinary activities in this type are substituting out-of-date documentation with up-to-date documentation, making semi-formal, say, in UML to document current program code, and updating the documentation with test plans.
Groomative	Ordinary activities in this type are substituting components and algorithms with more efficient and simpler ones, modifying the conventions for naming data, changing access authorizations, compiling source code, and doing backups.
Preventive	Ordinary activities in this type perform changes to enhance maintainability and establish a base for making a future transition to an emerging technology.
Performance	Activities in performance type produce results that impact the user. Those activities improve system up time and replace components and algorithms with faster ones.
Adaptive	Ordinary activities in this type port the software to a different execution platform and increase the utilization of COTS components.
Reductive	Ordinary activities in this type drop some data generated for the customer, decreasing the amount of data input to the system and decreasing the amount of data produced by the system.
Corrective	Ordinary activities in this type are correcting identified bugs, adding defensive programming strategies and modifying the ways exceptions are handled.
Enhancive	Ordinary activities in this type are adding and modifying business rules to enhance the system's functionality available to the customer and adding new data flows into or out of the software.

Tripathy and Naik (2015)



Building vs Maintaining Software

Visual Studio MAGAZINE

HOME NEWS TIPS & HOW-TO NEWSLETTERS W

VISUAL STUDIO VISUAL STUDIO CODE C#/VB .NET CORE XAMARIN/MOBILE TYPESCRIPT/BLAZ

PRACTICAL .NET f in t

In Praise of the Maintenance Programmer

The developers building new applications are very nice people, of course. But the real heroes of the programming world are the developers maintaining and extending existing applications.

By Peter Vogel 12/16/2014

Back in 1984, I was fresh out of school and ready to be hired as a developer. I was hired by a large multi-national corporation ... and immediately put on the maintenance team for an existing application. At the time, that decision seemed reasonable. In retrospect, it seems spectacularly stupid. Actually, "crazy" would be a better description.

Maintenance is much harder than new development.



<https://visualstudiomagazine.com/articles/2014/12/01/in-praise-of-the-maintenance-programmer.aspx>



What is actually done in software maintenance?



What is involved in Software Maintenance?

- Understanding the client (need to listen and translate into requirements)
- Understanding the existing code (often harder than you think)
- Refactoring the existing code (apply the guidelines for developing maintainable code)
- Extending the existing code (adding functionality)
- Working as a team
- Managing client expectations (deliver what the client wants to a high standard in time)
- Managing the maintenance process (same as development; need to be organised)



Maintenance as a Murder Mystery!

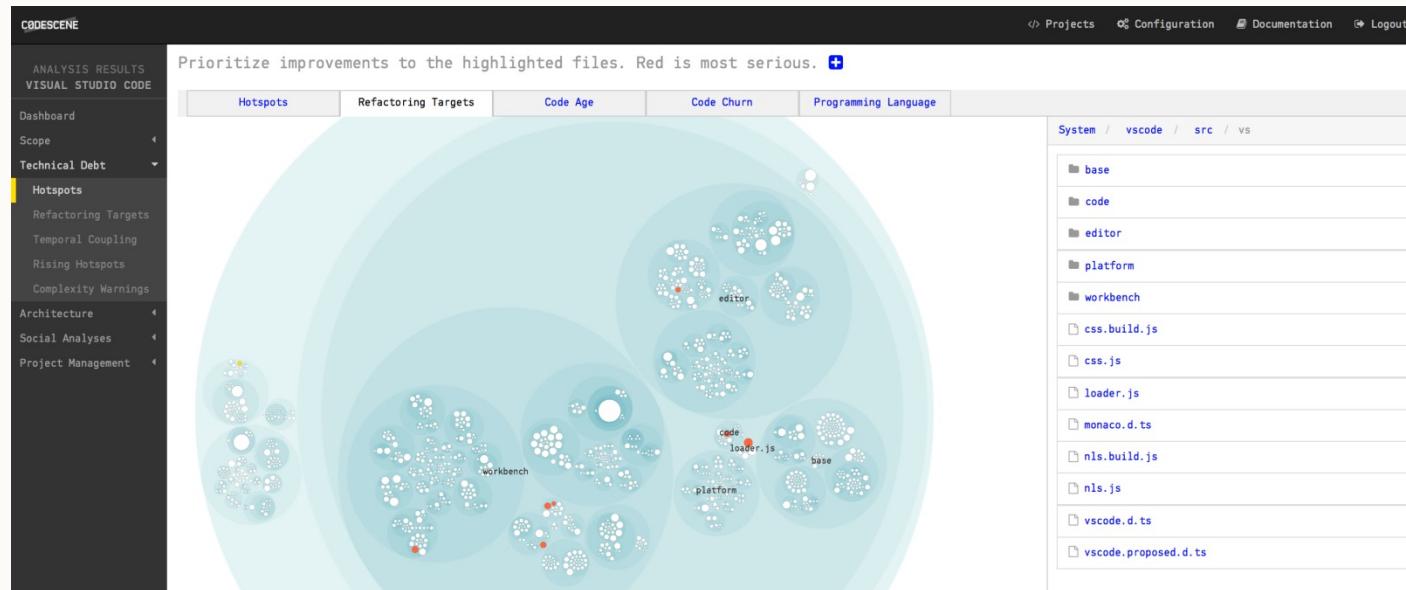
- Fixing a problem requires in depth knowledge of the crime scene, some specialist tools, and detective work!
 - Where did the crime take place?
 - Who committed the crime?
 - (not a witch hunt – but useful to know to try to locate related bugs)
 - Who and what else was involved?
 - Part detective, part programmer





Maintenance as a Murder Mystery!

- Some tools take this analogy a step further
 - e.g. CodeScene (<https://codescene.io/>)
 - Identifies patterns in the evolution of your code
 - Software forensics



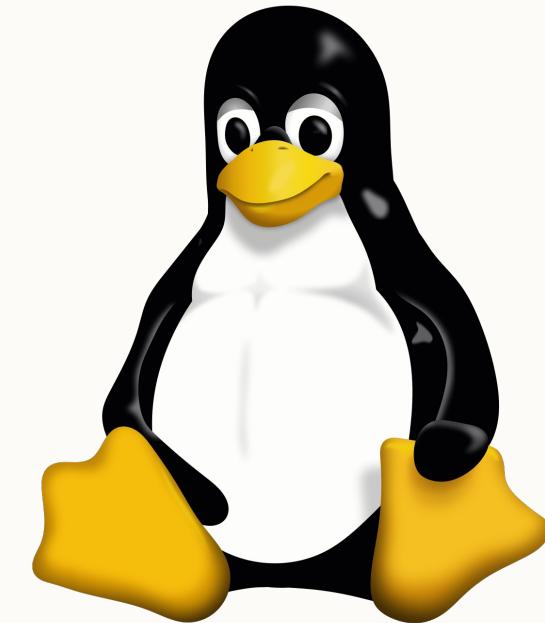
https://www.youtube.com/watch?v=n4P_I9rXKbE



How big is big? How complex is complex?

- Example: Linux Kernel

- How many lines of code?
 - 15 million lines of code (204.5 for a full Linux distribution)
- How many patches per release?
 - Approx. 10,000 patches in each release
 - Releases every 2-3 month
- How many developers?
 - Each release by 1,000s of developers
 - 200 corporations

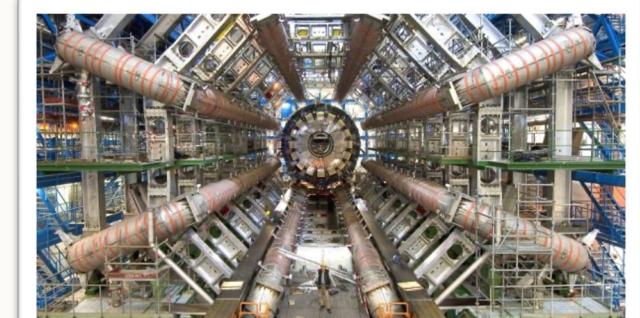




How big is big?

- Curiosity: 2.5 million lines
- CERN: 3 million lines (+50 million in projects)
 - Large hadron collider (particle accelerator)
- Windows: 50 million lines
- The code that guided the Apollo 11 module in 1969: 100,000 lines

- Of course, it is not just lines of code
 - 4000 people worked on the code base of Windows 10





Your New Job

- You've just started in a small company as a developer in a small team
- The company has just acquired a new software system
- You have been tasked with understanding and working with the code
- What do you do?





Your New Job

- Some suggestions from last year's cohort
 - Try to understand the code
 - How do you do it?
 - Look for the documentation
 - Let's hope the person who has written the code base has done the documentation!
 - Talk to team members
 - Scary thought? Need to learn to communicate
 - This is one of the key objectives of the group project!



What is involved in Software Maintenance?

- Understanding the client
- Understanding the code
- Refactoring the code
- Extending the code
- Working as a team
- Managing client expectations
- Managing maintenance process



What is involved in Software Maintenance?

- Understanding the client
- Understanding the code (1a/b)
- Refactoring the code
- Extending the code (2)

- Working as a team
- Managing client expectations
- Managing maintenance process (3)



1a. Making Sense of System Structure

- There could be hundreds or thousands of source code files in a project
- Program comprehension accounts for 50% of the total effort expended throughout the life cycle of a software system (Tripathy and Naik 2015).
- How to make sense of them?
 - It makes sense to look at relationships between classes
 - To do this we can use visualisation techniques
 - Understanding OOP is critical here





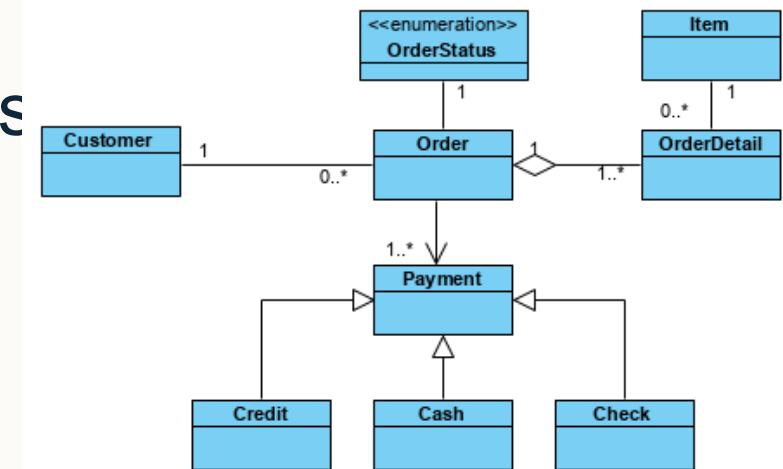
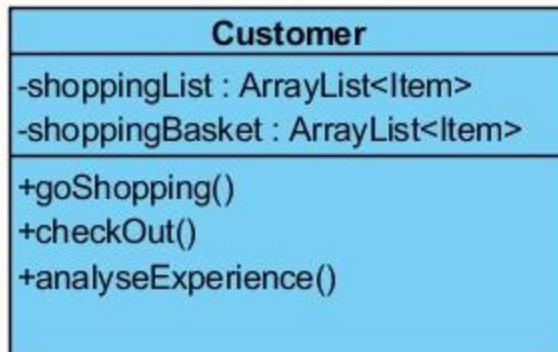
1a. Making Sense of System Structure

- Class Diagrams

- Show a set of classes and their relationships
- Addresses static design view of a system

- Classes

- Blueprints (templates) for objects
- Contain data/information and perform operations

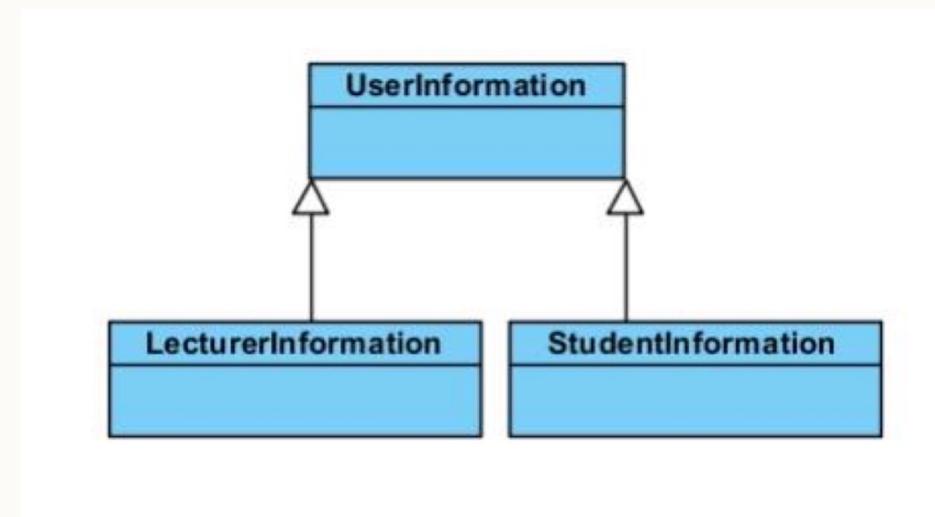


What is in the three compartments?



1a. Making Sense of System Structure

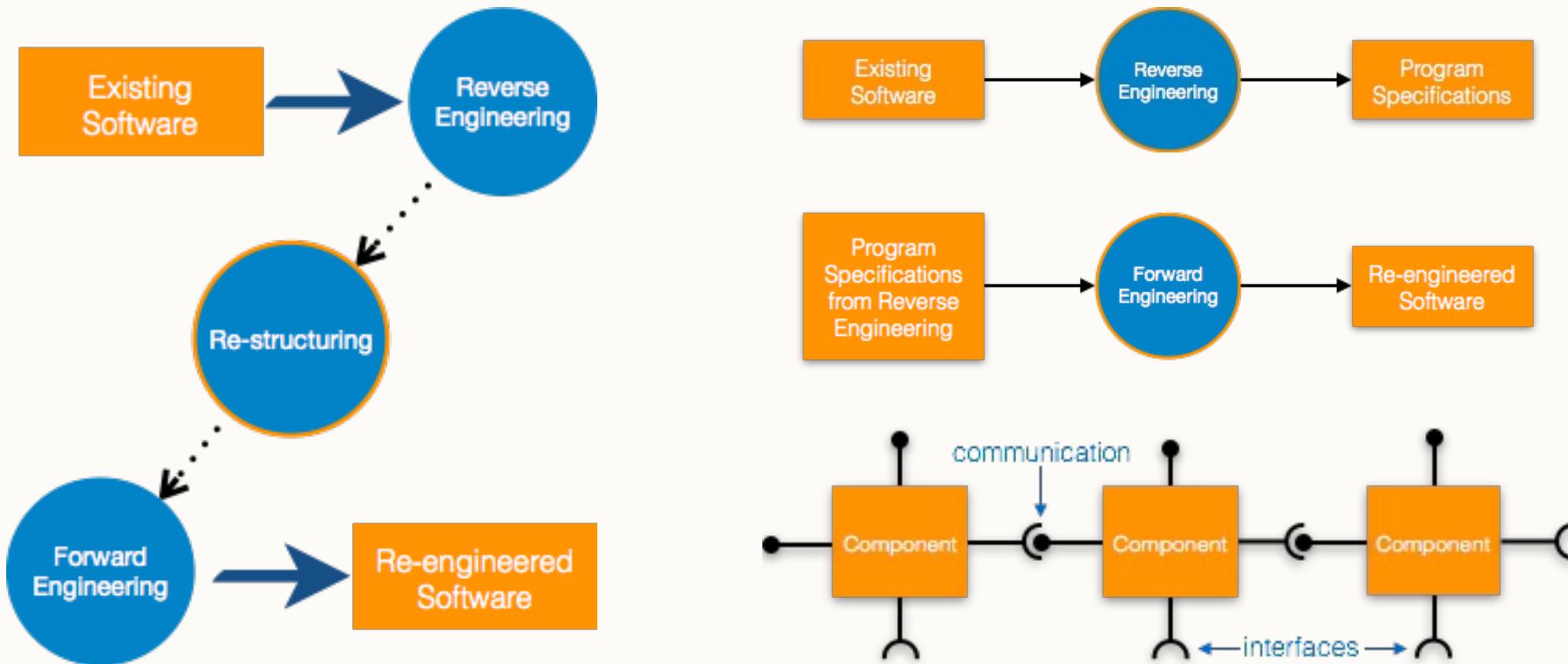
- Class diagrams offer a great way of showing inheritance and polymorphism



Which of the two do we see in this diagram?



1a. Reverse Engineering



https://sceweb.sce.uhcl.edu/helm/WEBPAGES-SoftwareEngineering/myfiles/TableContents/Module-13/software_maintenance_overview.html



1a. Reverse Engineering

- Transform these classes into a simple class diagram and do some maintenance

```
1 package ZooSystem;  
2  
3 public class Piranha extends Fish{  
4 }
```

```
1 package ZooSystem;  
2  
3 public abstract class Bird extends Animal{  
4 }
```

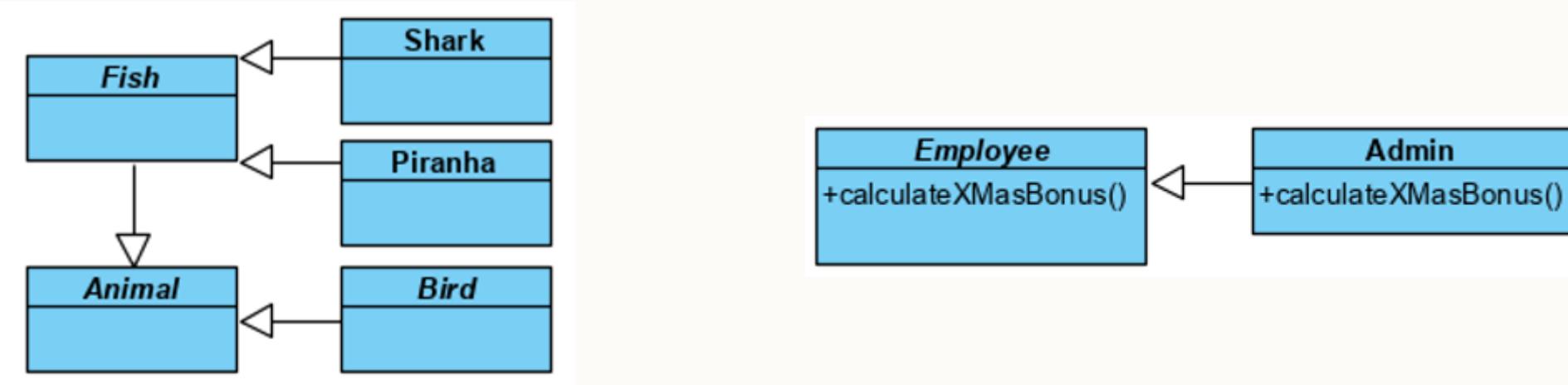
```
1 package ZooSystem;  
2  
3 public class Shark extends Fish{  
4 }
```

```
1 package ZooSystem;  
2  
3 public class Admin extends Employee{  
4  
5     @Override  
6     public int calculateChristmasBonus(){  
7         int bonus=(int)((double)getSalary()*0.08);  
8         return bonus;  
9     }  
10  
11     public Admin(String name){  
12         super();  
13         setEmployeeName(name);  
14     }  
15 }
```



1a. Reverse Engineering

- Simple class diagram (only including relevant methods)





1a. UML in Practice

- Class diagrams can help sometimes (but not always)
- They summarise the content of classes and relationships between them in the simplest possible way
- Does it make sense to build a detailed *class diagram* for an entire BIG project?
 - There are other methods to do this and other UML techniques to look at software at a higher level (e.g. deployment diagrams)
- Code can be generated directly from UML for maximum consistency



013 </>

1a. UML in Pract

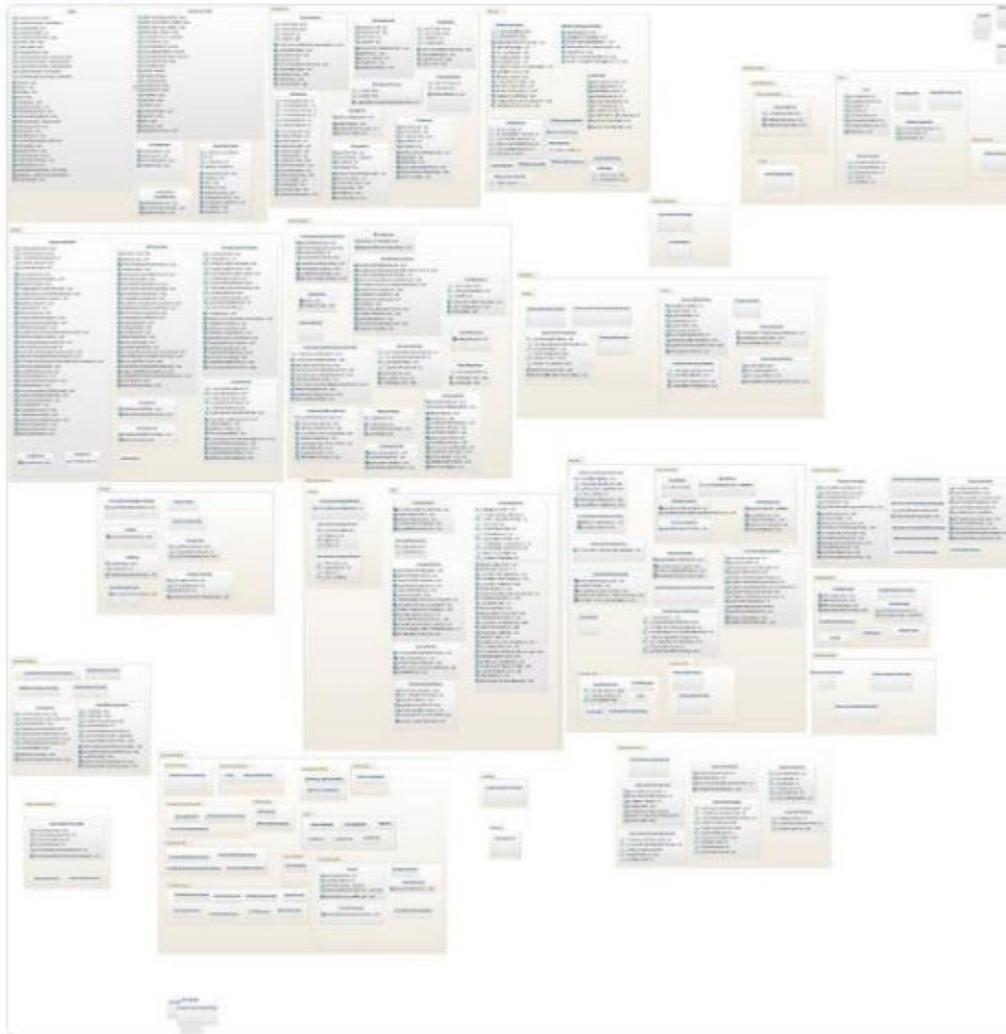


GenMyModel
@GenMyModel

[Follow](#)

- Caution! This is still a relatively small number of classes
- Use the diagrams sensibly, where they can help
- In this case using a high-level class diagram would be advisable

Wow! User rajiv's #UML class diagram is one of the biggest we've seen! Impressive!
goo.gl/lI5dIY





1b. Understanding the code itself

- As well as understanding the structure, we need to understand what the code actually does
- We need to understand the code itself to make the right decisions for producing robust and maintainable code



1b. Understanding the code itself

- Let's consider adding a method to set an employee phone number
 - Which class should it go in?
 - In Admin, or the abstract class Employee?
 - Should it be public or private?
 - What variable will you use?
 - What type?
 - What if another subtype of Employee already has a telephone method?
 - And what if one doesn't have a fixed telephone (**mobile number is share by a number of employees?**)

```
1 package ZooSystem;  
2  
3 public class Admin extends Employee{  
4  
5     @Override  
6     public int calculateChristmasBonus(){  
7         int bonus=(int)((double)getSalary()*0.08);  
8         return bonus;  
9     }  
10    public Admin(String name){  
11        super();  
12        setEmployeeName(name);  
13    }  
14}  
15}
```



1b. Following conventions vs being smart

- What does this C snippet do? Could you extend it easily?

```
float ██████████( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;
    i = 0x5f3759df - ( i >> 1 );
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );
    y = y * ( threehalfs - ( x2 * y * y ) );

    return y;
}
```

- No comments
- Poor variable names
- Obscure coding
- Redundant commenting out



Actually, it's Quake3's Fast InvSqrt()



Tonight, get out your last piece of Java coursework from Y1, and see if you still understand what you did 6 month ago

Not sure about this.



1b. Following conventions vs being smart

- Actually, there were some comments – *how helpful are these?*

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

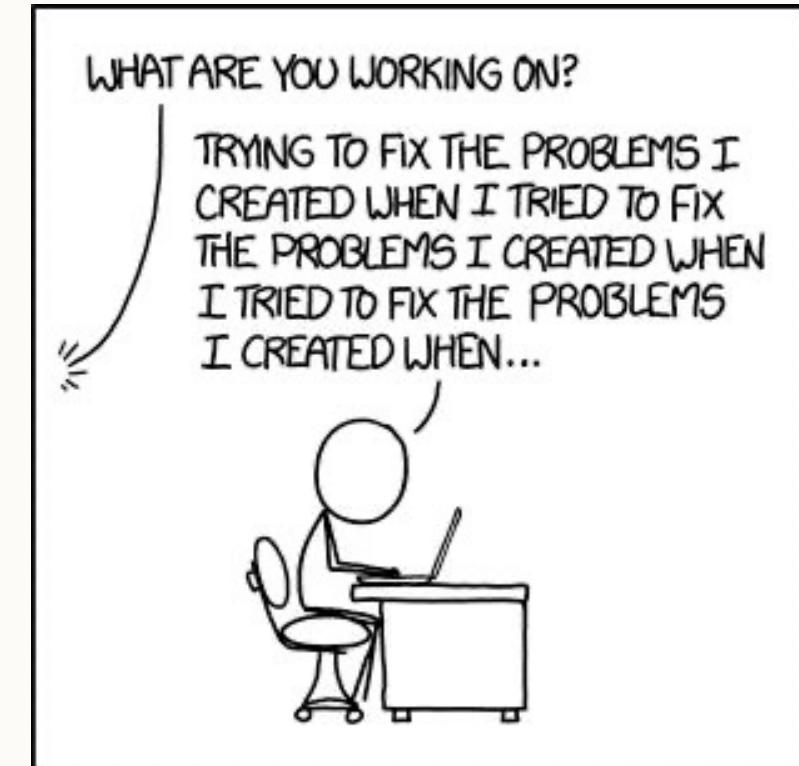
    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;                                // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 );                      // what the f***?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );          // 1st iteration
// y = y * ( threehalfs - ( x2 * y * y ) );          // 2nd iteration, this can be removed

    return y;
}
```



2. Extending Existing Code

- When you change existing code you risk breaking it
 - How do you know you haven't broken it?
 - What kind of testing do you need?
- Regression testing
 - A type of software testing to confirm that a recent program or code change has not adversely affected existing features
 - Running all your unit tests again





3. Software Maintenance Management

- This involves the management of people and not the management of conditional statements or linked lists
- The most under-appreciated part of our course, yet deep understanding of these processes often leads to successful future employment
- You are no longer the "Lone Coder" – from now on we encourage you to think like a pro and write your code as if someone else will need it
- You have looked at Git and version control in Y1; we will be going over it again ... and again ... and again ... until you use it by habit.
- Understanding how open source and software licensing works



3. Software Maintenance Management

- Use established processes to manage the design and implementation of changes
- Our Industrial Steering Group wants you to learn about how to actually perform agile development ... not just to know the word
 - Team meetings and a design process – e.g. Scrum
 - Team communication
 - Source code management (e.g. GIT)
 - Server-side testing (continuous integration server)
- This means talking to other people ...





Some final remarks



Polls

- See what they understood about the mod



Questions?





References

- Visser et al (2016) Building Maintainable Software - Java Edition
- Tripathy and Naik (2015) Software Evolution and Maintenance
- These slides have been adapted and reused from 2022/2023 – Thank you Peer and Robert (Bob)

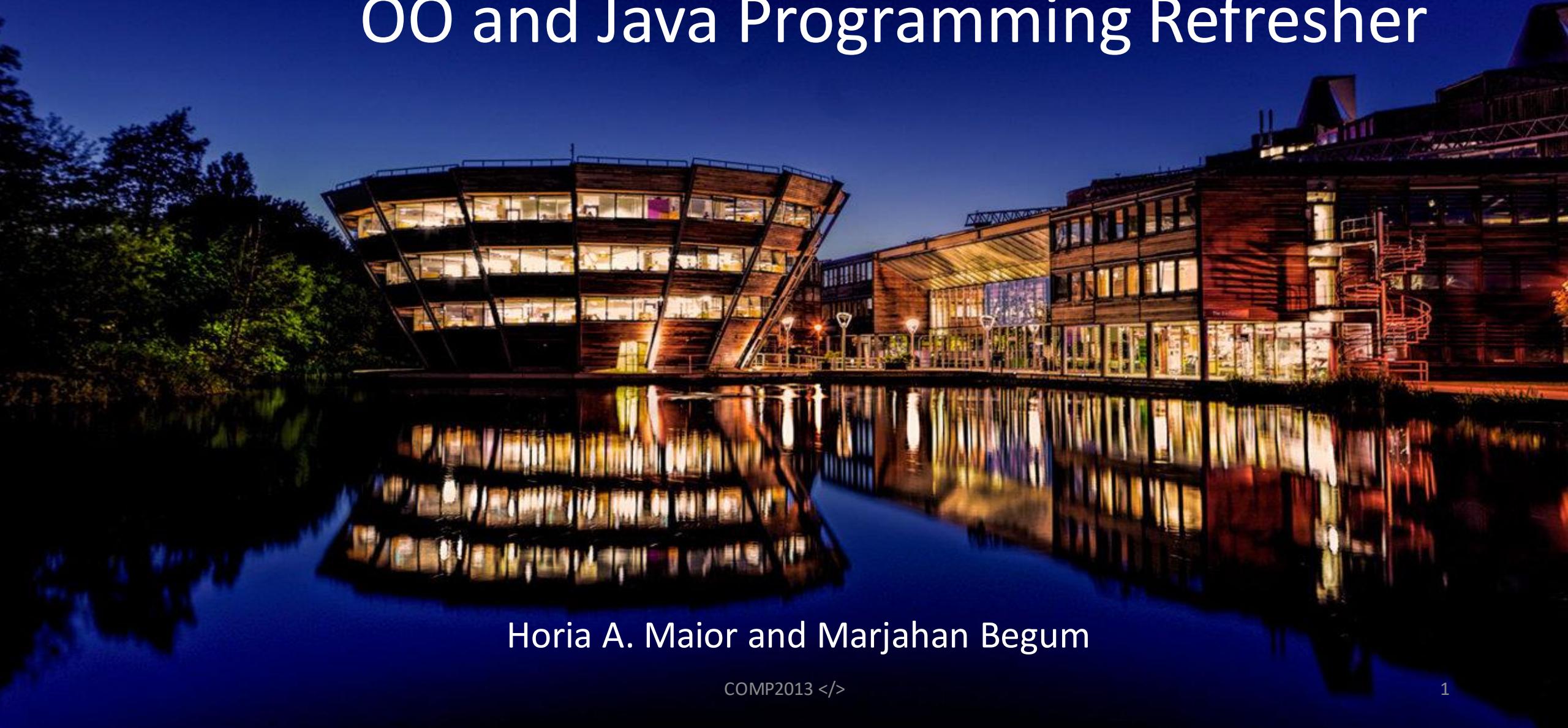


The University of
Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

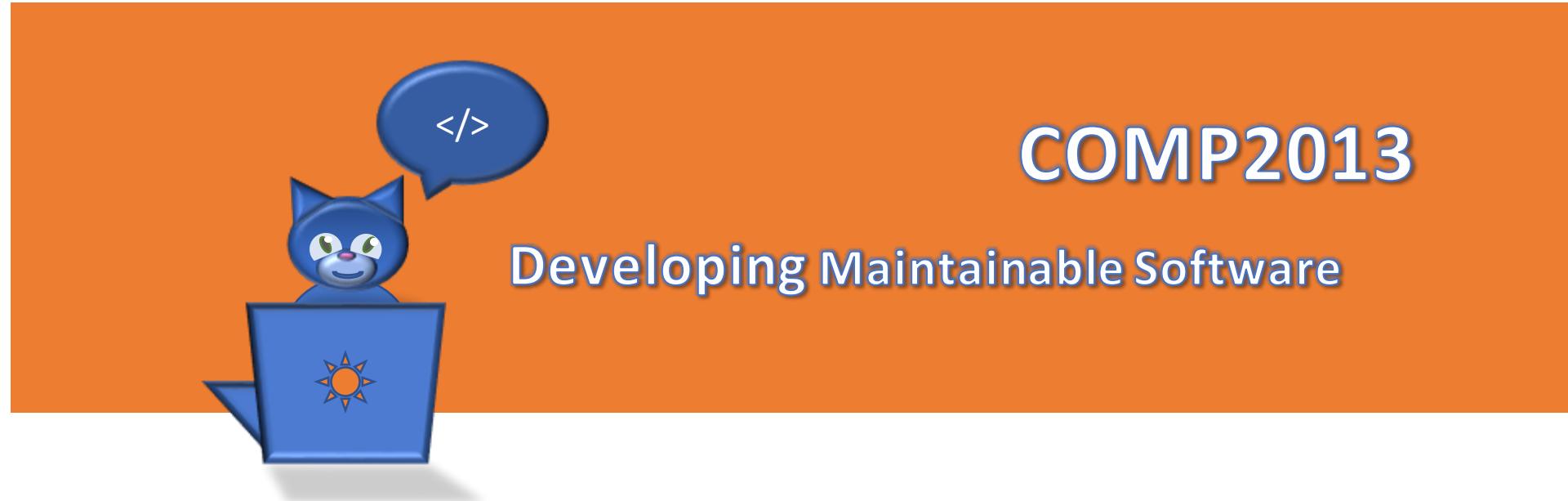
Lecture 01B

OO and Java Programming Refresher



Horia A. Maior and Marjahan Begum

</>



Lecture 01B (2023)

OO and Java Programming Refresher (1/2)

Horia A. Maior and Marjahan Begum

Topics for this Week

</>

- Lecture 01A
 - Welcome to the module
 - Developing maintainable software
 - Why COMP2013?
 - Challenges of software maintenance
- Lecture 01B
 - Labsheet
 - OO concepts & Java programming refresher (we complete this next week)
- Lab 01 – extra support in the lab
 - IntelliJ basics
 - Practicing Java basics
 - Working with existing code

Integrated Development Environment – IDEs

</>

- IntelliJ IDE, JetBrains

Fleet
Next-generation IDE by JetBrains
[Explore Fleet](#)
[Download](#)

IntelliJ IDEA
2023.2.2
The Leading Java and Kotlin IDE
[Explore IntelliJ IDEA](#)
[Download](#) [Buy](#)

TeamCity
2023.05.4
Powerful Continuous Integration out of the box
[Explore TeamCity](#)
[Download](#) [Buy](#)

Kotlin
Statically typed programming language for the JVM, Android and the browser
[Try](#) [Explore Kotlin](#)

IntelliJ IDEA for Education
2022.2.2
IntelliJ IDEA Community Edition with the built-in functionality for learning and teaching Java, Kotlin, and Scala
[Learn more](#)

JetBrains Academy
A plugin that adds educational functionality to your JetBrains IDE
[Try now](#)

Qodana
The code quality platform for your favorite CI tool
[Explore Qodana](#)
[Try now](#)

Aqua
A powerful IDE for test automation
[Explore Aqua](#)
[Download](#)

Filters [Reset](#)

Languages Java C/C++ C# Dart DSL F# Go Groovy HTML Java JavaScript, TypeScript Kotlin

Technologies

Product Type

</>

Lab sheet

- Fridays from 3-5
- IntelliJ IDE
- Different IntelliJ versions
 - Ultimate: Paid + much more functionality compared to community edition
 - Community: Free + open source
 - Educational: Community + one can practice programming with it
- Using Teams
 - Please use the "Questions" channel for help requests



- Evolution of IDEs and JDKs
 - There are many different ways of doing things and IDEs are evolving every day; no one can stay on top of this, unless perhaps you are a professional software developer
 - In this module we will teach you the ways that we have found work best
 - We are not claiming they are the best ways, but they work
 - Feel free to search the internet and learn some alternative best practices from the professionals

Lab 01 Overview

</>

LAB 01: INTELLIJ, AND "HELLO COMP2013 WORLD"

Content:

1. Preparation
2. Get familiar with IntelliJ
3. Implement a simple object-oriented example
4. Working with existing code

Everyone has different levels of experience with IDEs and OO programming. Jump in at the section, which suits your abilities.

NB: We do not expect everyone to solve all the "Challenges" listed below immediately. If you could not solve them now, you might want to consider coming back to them later in the module, once you have a bit more experience with the IDE and Java.

Please note that we have used a Windows environment to prepare this task sheet. If you are using a Linux or Mac environment and get stuck, please ask us, and we will try to help.

PART 1: PREPARATION

</>

To maintain software, you need to be familiar with modern Integrated Development Environments (IDEs). As we said in the first lecture, you should use the IntelliJ IDEA IDE¹ in this module. As for the programming language, everything from Java 12 onwards is suitable, but we recommend that you use one of the latest Java Development Kits.

Here are some tips for installing the JDK and IDE on your private machine:

- Java JDK (**not JRE**)
 - Download open-jdk GA release (<https://jdk.java.net/19/>)
 - Difference between GA and Reference release:
<https://stackoverflow.com/questions/50316397/difference-between-openjdk-10-and-java-se-10-reference-implementation>
 - System environment > Set this up in System Variables (bottom) rather than User Variables (top) if it is your own machine
 - JAVA_HOME > Path to JDK folder
 - Path > add "%JAVA_HOME%\bin"
 - Check that java installation works, using "java -version"
 - If it shows Java 8, you can use "where java" to check out why
 - Remove "C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe" or move the java path variable ("%JAVA_HOME%\bin") to the top of the list
- IntelliJ IDEA IDE
 - Download IntelliJ Community Edition: <https://www.jetbrains.com/idea/download/>
 - Run IntelliJ installer > Choose: 64 bit launcher + associate .java with IntelliJ + update context menu (if you want)
 - Install and then run IntelliJ > need to agree to terms > IntelliJ should open

If you encounter any issues when installing the IDE and JDK on your local machine, let us know, and we are trying to help you to fix those issues.

PART 2: GET FAMILIAR WITH INTELLIJ

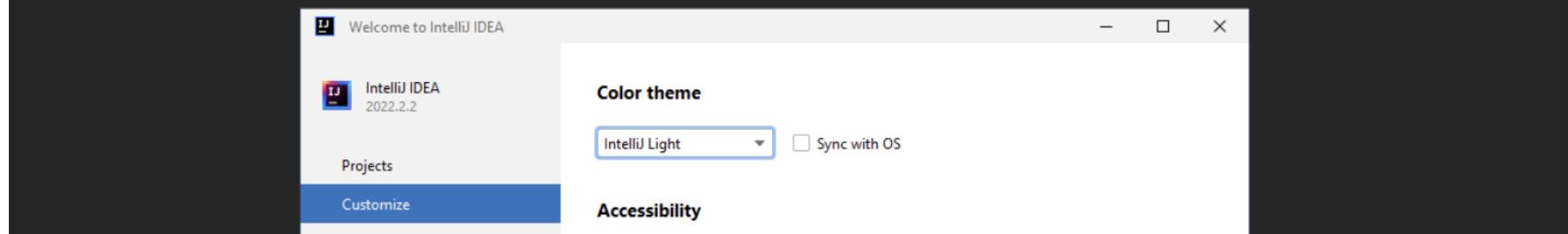
</>

The following information will guide you through programming a simple "Hello COMP2013 World" program in IntelliJ. Feel free to skip this if you know how to set up a new project in IntelliJ, using Java 12+ and write "Hello COMP2013 World" to the console.

One of the things introduced in Java 9 is formal modularity, in order to create structure inside large software systems. The new hierarchy in a Java project is as follows: Module > Package > Class > Fields and Methods. In one of the later lectures, we will talk about the overall concept in more detail. For now, we only need to be aware that this concept exists, as we should have one additional file (module-info.java) in the new projects we create.

For the following demonstration, I will use IntelliJ IDEA 2022.2.2, which is the version installed on my machine. The university machines and the Virtual Desktop (VD) have IntelliJ IDEA 2022.1.3 installed, but the difference between the two versions is not drastic.

- Start IntelliJ. If you are using the VD, the easiest way to start IntelliJ IDEA is to type "intelliJ" into the windows search box.
- In the "Welcome to IntelliJ IDEA" screen choose {Customize > All settings ...} and in the coming screen {System Settings > Default project directory} and choose your workspace directory, where you want to save your projects in. Always make sure you are using your personal network storage (I used "C:\Workspace\Java\IntelliJ", but yours will be different).



PART 3: IMPLEMENT A SIMPLE OBJECT-ORIENTED EXAMPLE

Create a new Java project, name it `BikeProject`, and add a new package `com.theBestBikeShop` to the `src` folder. Create a `module-info.java` file and add relevant information ([opens com.theBestBikeShop](#)). Download the code `Bicycle.java` from Moodle and import it into your project. To do this, drag and drop the file in the `src\com.theBestBikeShop` package, and then click `{Refactor}` in the appearing popup.

- If you want to check, where the `Bicycle.java` file is located, right click the file in the Project Explorer and select `{Copy Path > Reference...}`. In the popup, you can see the absolute path.

You can compile the project using `{Build > Build Project}` and there should be no errors, but it does not do anything, as it does not have a `main()` method yet.

Task: Extend the bike project.

- Create a `BikeApp` class including the `main()` method which instantiates a `Bicycle` object. A neat trick to create the `main()` method is to type `main` and use `{TAB}` for auto-complete.
- Write a line of code to change the speed of the bicycle you just created.
- Write a constructor for the `Bicycle` class, so that you can set the initial gear and speed values. This will create an error. Go to the `BikeApp` class and change it, so that the new constructor is used for generating the `Bicycle` object
- Add a method called `switchLightStatus` to the `Bicycle` class that turns the light on if it is off, and vice versa. Add a method `currentState` to the `Bicycle` class to print out the current speed, gear and light state in the console. Test the functionality of both added methods by adding some code to the `BikeApp` class



Let's suppose we also want a class for a mountain bike, which has specific features associated with it. It makes sense to create a subclass of a [Bicycle](#) class, i.e. inherit from it.

In the [Project Explorer](#), right click your `com.theBestBikeShop` package and select [{New > Java Class}](#). Name the new class `MountainBike` and add "... extends Bicycle" to the class signature. Then hoover with the mouse over the appearing red line and pressed [{Alt + Shift + Enter}](#) on the keyboard, which automatically created the required constructor.

Hey presto, your subclass is created.

We will cover more about inheriting from classes in the coming lectures. But for now, let's just add two Boolean variables to store whether a bike has a front suspension and a rear suspension.

- Modify the constructor of the `MountainBike` class to take in value for the two suspension variables, and set them in the constructor. Add a method called `isFullSuspension`, which returns `true` only if the bike has both front and rear suspension. In the `BikeApp` class, create objects for two different mountain bikes, one with full suspension, and one without.
- Override the `currentState()` method in the `Bicycle` class from within the `MountainBike` class, providing gear, speed, light status, and suspension state information. Test the functionality of the added method by adding some code to the `BikeApp` class.

Challenge: Automatically generate getters, setters, and constructors.

The IntelliJ IDE menu [{Code}](#) contains a lot of functionality for automating processes such as writing getters and setters and producing constructors. Delete all existing getters, setters, and constructors within your bike source code and use [{Code > Generate ...}](#) to get back to how the code looked before.



PART 4: WORKING WITH EXISTING CODE

In this part we will download and explore a larger existing codebase. To do this we will learn some tips for navigating project code. Please note: The following is written for Eclipse, but will work similarly in the IntelliJ IDE.

Challenge: Try to work out how it works in IntelliJ. Some advice is provided in orange.

Let's have a look at some real world source code: a game called "Diamond Hunter". You can find more information about this game here: <https://www.youtube.com/watch?v=AA1XpWHhxw0>.

First, you need to download the project zip file DiamondHunter.zip from Moodle, unzip it, and import the resulting folder into IntelliJ. In IntelliJ IDEA, just open the project. You might have to define the Project SDK. Use {File > Project Structure > Project Settings > Project > Project SDK} and choose the one you want to use from the pull-down menu] [If you want a real challenge, download DiamondHunterSrc.zip instead and set up your own project in IntelliJ IDEA.

HOW DO YOU FIND A SPECIFIC CLASS (OPEN A CLASS BY NAME)?

Let's assume we want to find the class GameState. An efficient way is to choose In IntelliJ use {Navigate > Class...} the class GameState that is listed. This class will then appear in the editor window.

HOW DO YOU GET AN OVERVIEW OF METHODS AND FIELDS OF A CLASS?

Use {View > Tool Window > Structure}

HOW CAN YOU SEE THE INHERITANCE/TREE OF A SPECIFIC CLASS?

Lab – Looking ahead

</>

- Error Handling and Debugging
 - Hoover over error
 - Click on the line and the appearing red light bulb
 - Compile the code, ignoring the warning, and you get a list of errors
 - Following up errors > always start reading them from the bottom to the top
 - You need to try to solve fix the code yourselves!!
 - Talk to a rubber duck!
 - Search the web
 - You cannot just ask us without you spending enough time on it. This is how you learn!



Project

- BikeProject ~/Software Development/COMP2013/BikeProject
 - .idea
 - out
 - src
 - com.horiamaior
 - Bicycle
 - BikeApp
 - MountainBike
 - .gitignore
 - BikeProject.iml
- External Libraries
- Scratches and Consoles

BikeApp.java x Bicycle.java

```
1 package com.horiamaior;
2
3 public class BikeApp {
4     public static void main(String[] args) {
5         //Bicycle bike = new Bicycle(11,2);
6         bike.speedUp(10);
7     }
8 }
9
10
11
12
```

! Create local variable 'bike'
! Create field 'bike' in 'BikeApp'
! Create parameter 'bike'
! Rename reference
Try to resolve class reference

Press F1 to toggle preview

Build Build Output x

BikeProject: build failed At 27/09/2023, 10:27 with 2 errors 5 sec, 771 ms

BikeApp.java src/com/horiamaior 2 errors

! cannot find symbol variable bike :6
! cannot find symbol variable b1 :7

/Users/psahm2/Software Development/COMP2013/BikeProject/src/com/horiamaior/BikeApp.java:6:9
java: cannot find symbol
symbol: variable bike
location: class com.horiamaior.BikeApp

oo concepts & java programming refresher

OO Design – Class Diagram and Initial Planning

</>

- Requires you to sketch out classes and methods to implement technical problems or real life objects.
- Think about Input, Output,
- Key Stakeholders,
- How things interact

OO Design - Problems/steps in this process

</>

- Handling Ambiguity
- Define Core Objects
- Analise relationships
- Investigate actions



≡ OUR ZOO ≈

PLAN YOUR
VISITGIFTS &
EXPERIENCES

LATEST NEWS

PREVENTING
EXTINCTION

BOOK YOUR TICKETS

OUR ZOO ABOUT US ANIMALS PLANTS & GARDENS ZOO MEMORIES SUPPORT US

MORE ▾

Home > Our Zoo > Animals

Meet our
≡ ANIMALS ≈Discover the beautiful animals & habitats at
Chester Zoo

SEARCH



ANIMAL GROUP



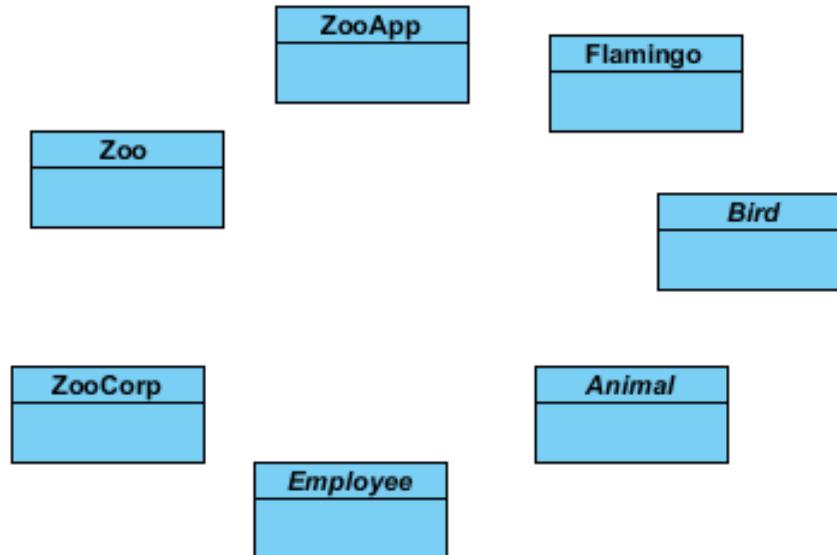
Step 1: Ambiguity

</>

- 6Ws
 - Who?
 - What?
 - Where?
 - When?
 - How?
 - Why?

Step 2: Define Core Objects

</>



Step 3: Analyse Relationships

</>

Step 4: Investigate Actions

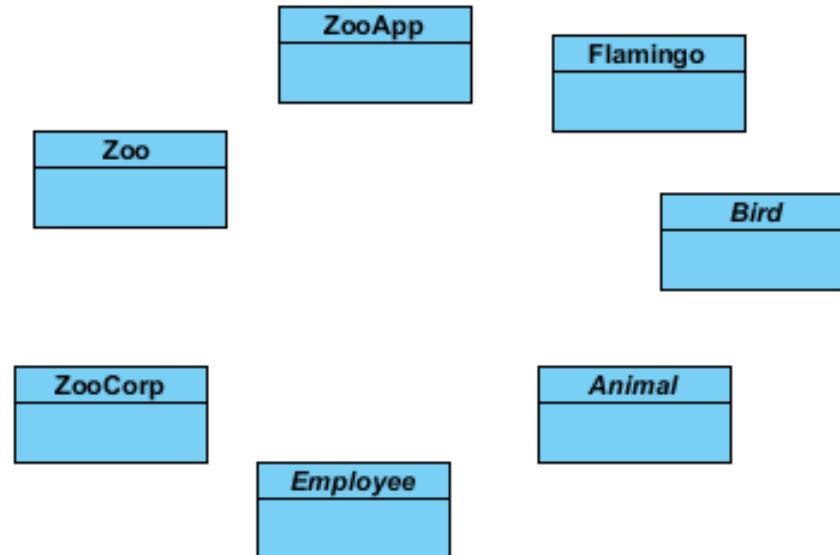
</>

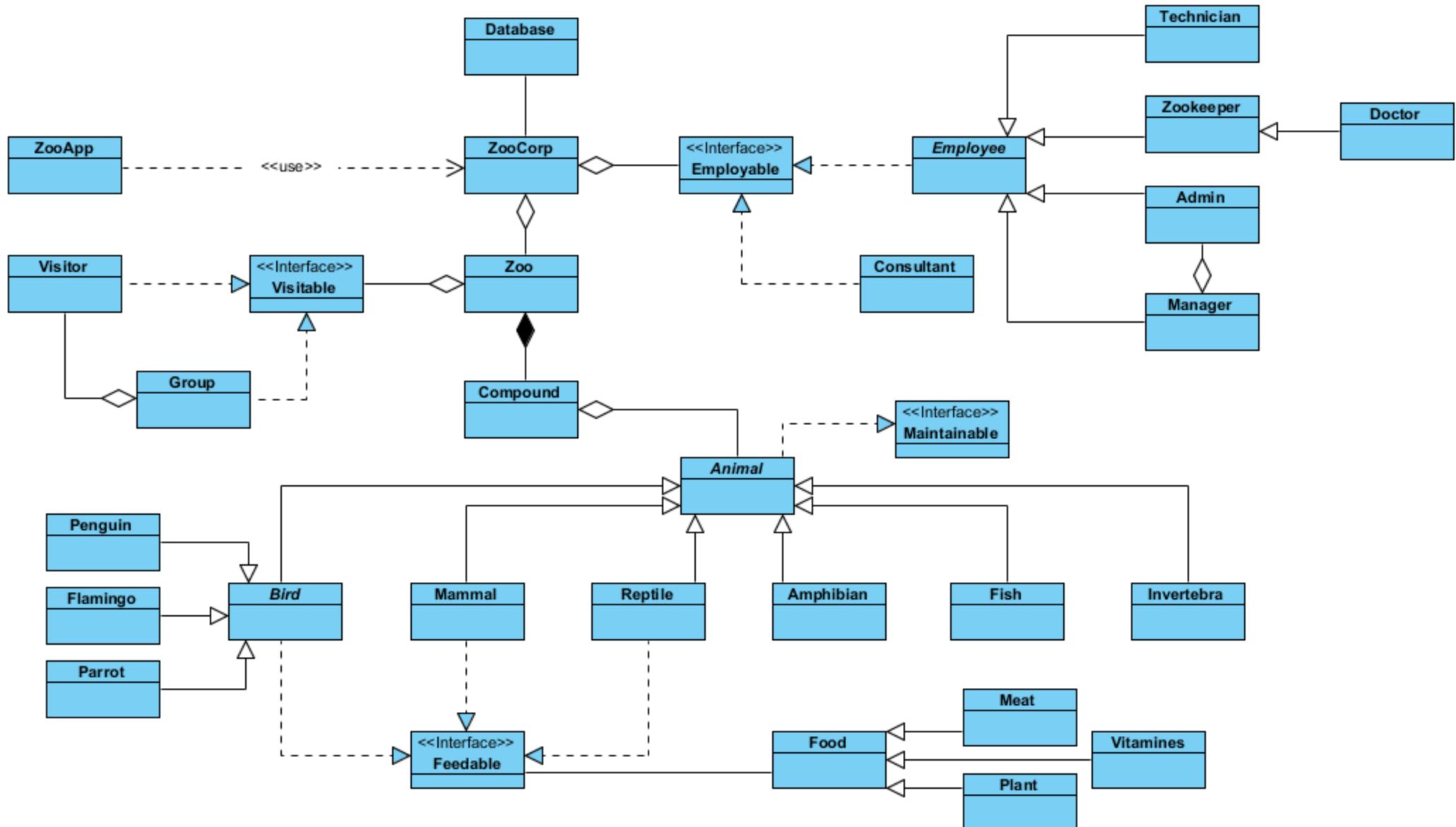
Case Study: Zoo Management

</>



- Come up with a draft class diagram for the Zoo Management problem
 - Note that this is only a small choice of relevant classes!





What assumptions have you made?

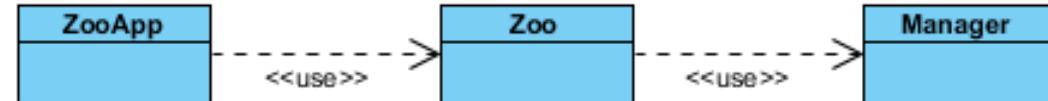
</>

- What were the assumptions that you made when designing this?
- How would it change if you worked with a customer?

Case Study: Zoo Management

</>

As we focus on Java basics today we want to keep it simple ...



ZooProjectStage1

Project

ZooProjectStage1 E:\Teaching\COMP2013-G52DMS\G52D

- .idea
- out
- src
 - com.Hagenbeck
 - Manager
 - Zoo
 - ZooApp
 - module-info.java
 - ZooProjectStage1.iml
- External Libraries
- Scratches and Consoles

```
1 package com.Hagenbeck;
2
3 public class ZooApp {
4
5     public static void main(String[] args) {
6         Zoo zoo=new Zoo();
7         System.out.println("Job done!");
8     }
9 }
10
```

⚠ 1 ⏪ ⏩

Project

ZooProjectStage1 E:\Teaching\COMP2013-G52DMS\G52D

- .idea
- out
- src
 - com.Hagenbeck
 - Manager
 - Zoo
 - ZooApp
 - module-info.java
- ZooProjectStage1.iml

External Libraries

Scratches and Consoles

Project ZooApp.java Zoo.java Manager.java

```
1 package com.Hagenbeck;
2
3 public class Zoo {
4
5     public void doSomething() { Manager manager=new Manager(); }
6
7 }
8
9 |
```

2

ZooProjectStage1 > src > com > Hagenbeck > Manager

ZooApp ▾

Project

Project

ZooProjectStage1 E:\Teaching\COMP2013-G52DMS\G52D

- .idea
- out
- src
 - com.Hagenbeck
 - Manager
 - Zoo
 - ZooApp
 - module-info.java
- ZooProjectStage1.iml

External Libraries

Scratches and Consoles

ZooApp.java × Zoo.java × Manager.java ×

1 package com.Hagenbeck;

2

3 public class Manager {

4 }

5 |

Structure

Favorites

TODO Problems Build Terminal

1 Event Log

 Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (2 minutes ago)

CUIVIPZU13 </>

5:1 CRLF UTF-8 4 spaces

ZooProjectStage1 > src > com > Hagenbeck > ZooApp

ZooApp ▾

Project

Run: ZooApp

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\lib\idea_rt.jar=50592:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\bin" -Dfile.
```

Job done!

Process finished with exit code 0

Structure

Favorites

Run TODO Problems Build Terminal

Event Log

 All files are up-to-date (moments ago)

5:1 CRLF UTF-8 4 spaces



COMPILED </>

UK | CHINA | MALAYSIA

- Object-oriented programming is founded on these ideas:
 - **Abstraction**: Simple things like objects represent more complex underlying code and data
 - **Encapsulation**: The ability to protect some components of the object from external access
 - **Inheritance**: The ability for a class ("subclass") to extend or override functionality of another class ("superclass")

- Object-oriented programming is founded on these ideas:
 - **Polymorphism**: The provision of a single interface to entities of different types
 - Compile time polymorphism: Method overloading
 - Run time polymorphism: Method overriding

Programming Refresher

</>

- Public vs. Private
- Accessors and Modifiers
- Encapsulation
- The "this" Keyword
- Constructors
- Passing by Value or Reference?
- More Hacks
- Static Fields and Methods

Public vs. Private

</>



- What are the general **access rules** for constructors, methods, helper methods, fields, and static constants?
 - Constructors and methods: Usually declared public
 - Helper methods that are needed only inside the class: Usually declared private
 - Fields: Usually declared private
 - Static constants: Usually declared public

Accessors and Modifiers

</>

- Accessors (also called Getters):
 - Methods that return values of private fields
- Modifiers (also called Mutators or Setters):
 - Methods that set values of private fields



ZooProjectStage2

Project

Project ▾ + - × ⚙ ZooProjectStage2 E:\Teaching\COMP20
➤ .idea
➤ out
src
➤ com.Hagenbeck
➤ Manager
➤ Zoo
➤ ZooApp
module-info.java
ZooProjectStage2.iml
➤ External Libraries
➤ Scratches and Consoles

ZooApp.java × Zoo.java × Manager.java ×

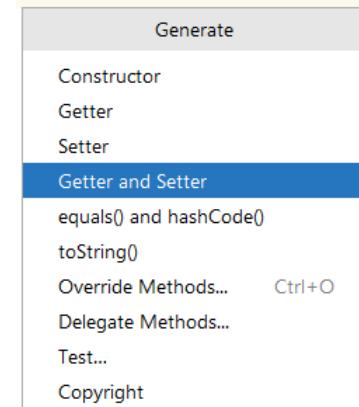
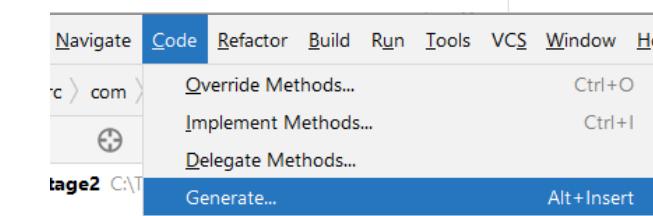
⚠ 1 ^ v

```
1 package com.Hagenbeck;  
2  
3 public class ZooApp {  
4  
5     public static void main(String[] args) {  
6         Zoo zoo=new Zoo();  
7         System.out.println("Job done!");  
8     }  
9 }  
10
```

Structure

Favorites

```
1 package com.Hagenbeck;
2
3 public class Zoo {
4
5     private String location;
6
7     public void doSomething() { Manager manager=new Manager(); }
8
9     public String getLocation() { return location; }
10
11    public void setLocation(String location) { this.location = location; }
12
13 }
```



Project

ZooProjectStage2 E:\Teaching\COMP20
> .idea
> out
src
 com.Hagenbeck
 Manager
 Zoo
 ZooApp
 module-info.java
 ZooProjectStage2.iml
> External Libraries
> Scratches and Consoles

Structure

Favorites

ZooApp.java | Zoo.java | Manager.java

```
1 package com.Hagenbeck;  
2  
3 public class Manager {  
4 }  
5
```



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage2 - Manager.java



ZooProjectStage2 > src > com > Hagenbeck > Manager



ZooApp



Project

Project



Run: ZooApp



```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\lib\idea_rt.jar=50606:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\bin" -Dfile.
```

Job done!

Process finished with exit code 0

Up

Down

Left

Right

Print

trash

Up

Down

Left

Right

Print

trash

Structure

Favorites

Run

TODO

Problems

Build

Terminal

Event Log

5:1 CRLF UTF-8 4 spaces

All files are up-to-date (moments ago)

UK | CHINA | MALAYSIA

Encapsulation

</>

- Hiding the implementation details of a class (making all fields and helper methods private) is called encapsulation
- Encapsulation helps with program maintenance: a change in one class does not affect other classes
- A client of a class interacts with the class only through well-documented public constructors and methods; this facilitates team development

The Keyword "this"

</>

- "this" refers to the implicit parameter inside your class
 - A variable that stores the object on which a method is called
 - Refer to a field
 - `this.field`
 - Call a method
 - `this.method(parameters);`
 - One constructor can call another
 - `this(parameters);`

Constructors

</>



- What is special about constructors (compared to other method types)?



- The constructor method is a special method of a class for creating object instances of that class
 - They often initialise an object's fields
 - They do not have a return type
 - All constructors in a class have the same name (the name of the class)
 - They may take parameters
- If a class has more than one constructor, they must have different numbers and/or types of parameters (constructor overloading)
- Important!
 - Java **provides** a default constructor for a specific class
 - If you define a constructor for a class, Java **does not provide** the default constructor any more

Constructors

</>

- Constructors of a class can call each other using the keyword "this" (referred to as constructor chaining) - a good way to avoid duplicating code
- Constructors are invoked using the operator new.
 - Declare a reference variable of the required type and then invoke the constructor method after the "new" keyword
- Parameters passed to "new" must match the number, types, and order of parameters expected by one of the constructors.



ZooProjectStage3

ZooProjectStage3 > src > com > Hagenbeck > ZooApp



Project

ZooProjectStage3 E:\Teaching\COMP20
> .idea
> out
src
 com.Hagenbeck
 Manager
 Zoo
 ZooApp
 module-info.java
 ZooProjectStage3.iml
> External Libraries
> Scratches and Consoles

ZooApp.java × Zoo.java × Manager.java ×

```
1 package com.Hagenbeck;  
2  
3 public class ZooApp {  
4  
5     public static void main(String[] args) {  
6         //Zoo zoo=new Zoo();  
7         Zoo zoo=new Zoo( location: "Hamburg");  
8         System.out.println("\nZoo:"+zoo);  
9     }  
10 }  
11 |
```

Favorites

TODO Problems Build Terminal

Event Log

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure...

11:1 CRLF UTF-8 Tab* ?

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage3 - Zoo.java

ZooProjectStage3 > src > com > Hagenbeck > c Zoo

ZooApp.java × Zoo.java × Manager.java ×

Project .idea out src com.Hagenbeck Manager Zoo ZooApp module-info.java ZooProjectStage3.iml External Libraries Scratches and Consoles

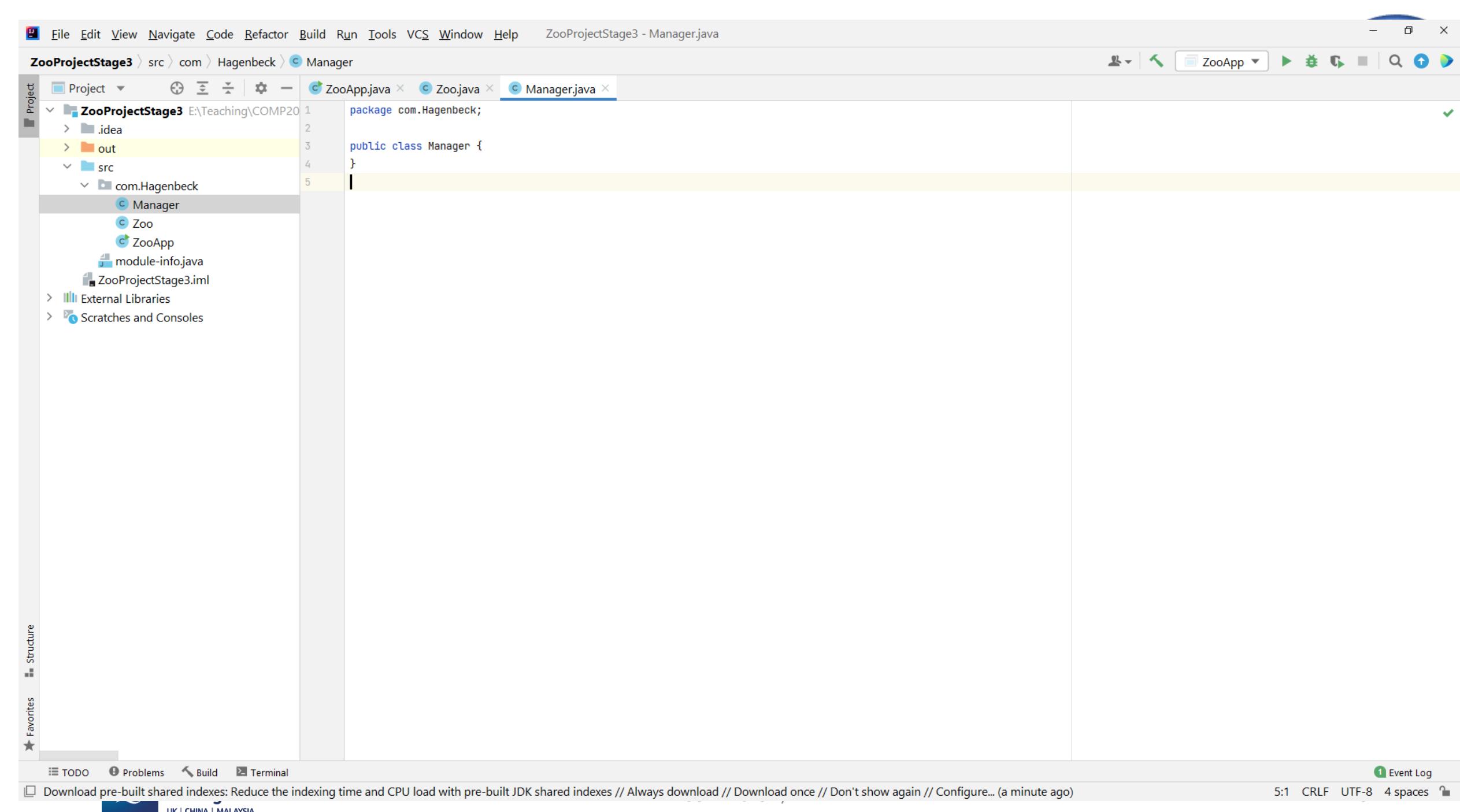
Structure Favorites

1 package com.Hagenbeck;
2
3 public class Zoo {
4
5 private String location;
6
7 public Zoo() { this(location: "Unknown"); }
8
9 public Zoo(String location) {
10 //this.location=location;
11 this.setLocation(location);
12 System.out.println("\nConstructing "+this);
13 }
14
15 public void doSomething() { Manager manager=new Manager(); }
16
17 public String getLocation() { return location; }
18
19 public void setLocation(String location) { this.location = location; }
20 @Override
21 public String toString() { return getClass().getSimpleName()+"("+this.getLocation()+")"; }
22 }
23
24

TODO Problems Build Terminal Event Log

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

33:1 CRLF UTF-8 Tab* UK | CHINA | MALAYSIA



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage3 - Manager.java

ZooProjectStage3 > src > com > Hagenbeck > ZooApp

Project ZooApp.java Zoo.java Manager.java

Run: ZooApp

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\lib\idea_rt.jar=50628:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\bin" -Dfile.encoding=UTF-8
```

Constructing Zoo(Hamburg)

Zoo:Zoo(Hamburg)

Process finished with exit code 0

Run TODO Problems Build Terminal

All files are up-to-date (moments ago)

UK | CHINA | MALAYSIA

Event Log 1

Some extra thought ...

Extras: Google/Facebook/etc OOP Interview Questions

</>

Exercise 1: Jukebox: Design a musical jukebox using object-oriented principles.

Exercise 2: Parking Lot: Design a parking lot using object-oriented principles.

Exercise 3: Online Book Reader: Design the data structures for an online book reader system.

Source: Cracking the Coding Interview 6th edition, by Gayle Laakmann McDowell

Constructors

</>

- What does the output look like?

```
public static void main(String[] args) {  
    Zoo zoo1;  
    zoo1=new Zoo( location: "Hamburg");  
    zoo1=new Zoo( location: "Munic");  
    Zoo zoo2=zoo1;  
    Zoo zoo3=new Zoo();  
    System.out.println("\nzoo1:"+zoo1);  
    System.out.println("Zoo2:"+zoo2);  
    System.out.println("Zoo3:"+zoo3);  
    zoo3.setLocation("Berlin");  
    zoo1.setLocation("Berlin");  
    System.out.println("\nzoo1:"+zoo1);  
    System.out.println("Zoo2:"+zoo2);  
    System.out.println("Zoo3:"+zoo3);  
    zoo1=new Zoo( location: "SanDiego");  
    System.out.println("\nzoo1:"+zoo1);  
    System.out.println("Zoo2:"+zoo2);  
    System.out.println("Zoo3:"+zoo3);  
}
```

Constructing Zoo(Hamburg)

Constructing Zoo(Munic)

Constructing Zoo(Unknown)

Zoo1:Zoo(Munic)

Zoo2:Zoo(Munic)

Zoo3:Zoo(Unknown)

Zoo1:Zoo(Berlin)

Zoo2:Zoo(Berlin)

Zoo3:Zoo(Berlin)

Constructing Zoo(SanDiego)

Zoo1:Zoo(SanDiego)

Zoo2:Zoo(Berlin)

Zoo3:Zoo(Berlin)



ZooProjectStage4

Acknowledgement

</>

- Slides based on material from
 - Bill Leahy's lecture slides
 - http://www.cc.gatech.edu/~bleahy/xjava/cs1311xjava05_poly.ppt
 - Maria Litvin's & Gary Litvin's book slides
 - <http://skylit.com/javamethods/ppt/Ch10.ppt>
 - Marty Stepp's lecture slides
 - <http://www.cs.washington.edu/331/>
 - Cracking the Coding Interview by Gayle Laakmann McDowell
 - And other lecturers delivering this module in the past ...

But I also contributed some stuff myself :-)

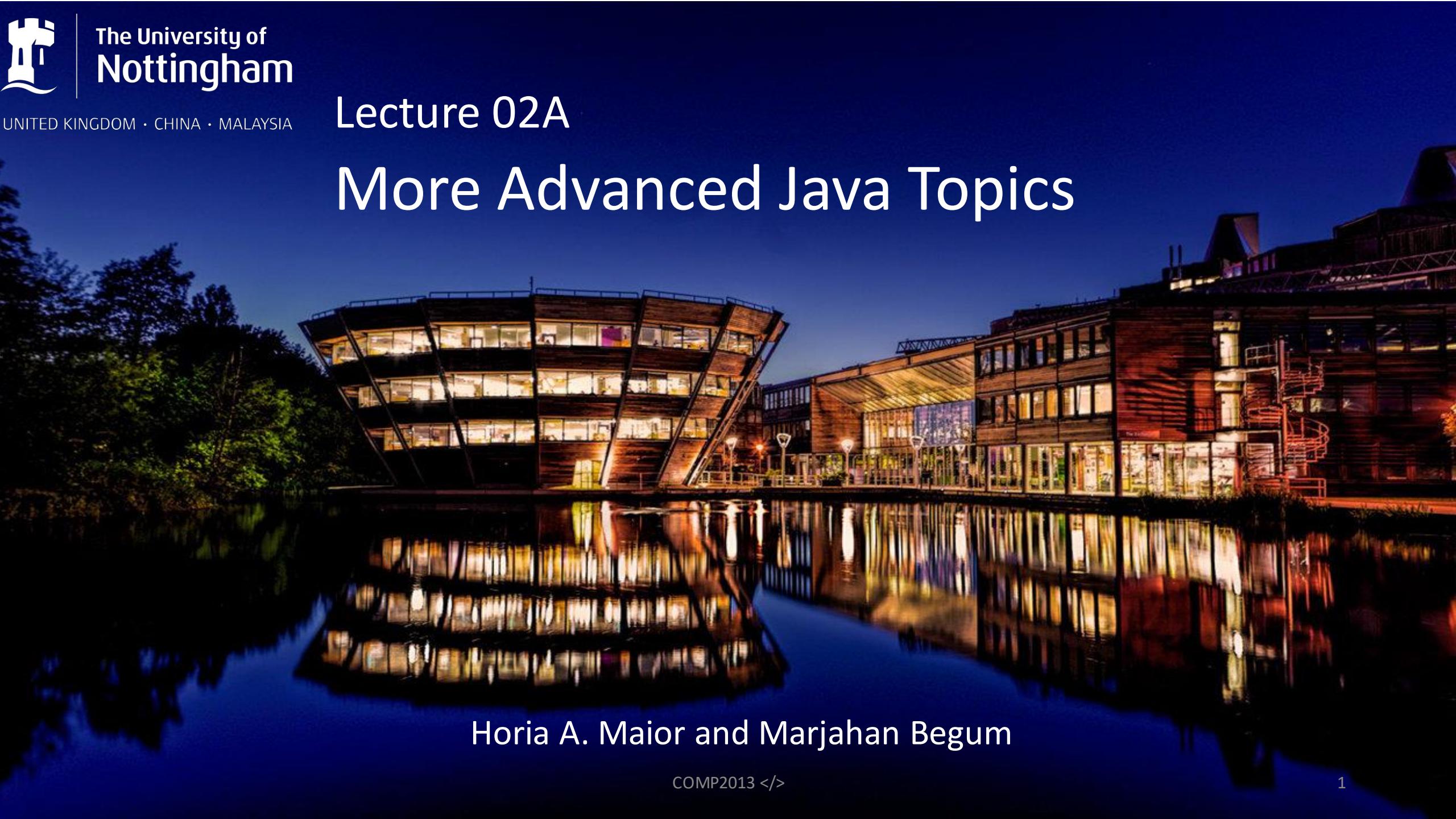


The University of
Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Lecture 02A

More Advanced Java Topics



A photograph of a modern university building at night, illuminated from within and reflected in the water in front of it. The building has a complex, angular design with multiple levels and large glass windows. The reflection in the water creates a symmetrical image of the building's lights.

Horia A. Maior and Marjahan Begum

</>

CD General Posts Files +

5 replies from Prathu, Robert, Joshua, and 2 others

Christopher Wainwright Monday 16:28

Marjahan Begum (staff) Monday 18:04 Edited

Dear all

Thank you for today's lecture. The slides are uploaded now. Apology for not uploading earlier. We will do it from next lecture.

See more

6 October 2023

Horia Major (staff) Friday 12:48

Lecture Recordings and Slides

Dear General all,

It was great to see you in class this week!

I just wanted to let you know that the lecture slides and recordings are available on moodle and in here:

- Lecture 01A: Introduction to Developing Maintainable Software [recording] [full slides]

See more

Developing Maintainable Software - Lecture Rec... personal >marjahan_begum_nottingham_ac_uk

Developing Maintainable Software Lecture 1B - O... personal >marjahan_begum_nottingham_ac_uk

Horia Major (staff) Friday 12:53

Dear General,

As a reminder, if you have completed the labsheet for week 1 you no longer need to come to the lab today. Please do come if you need support.

Best wishes,

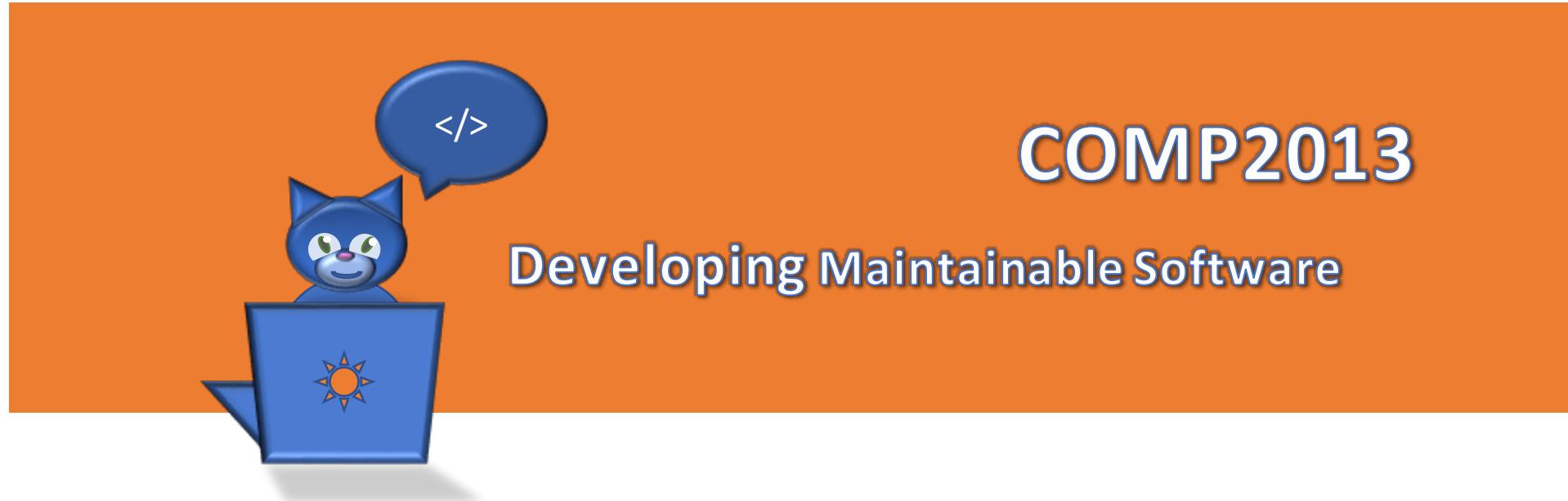
See more

New conversation

COMP2013 </>

</>

Please note that the slides published AFTER the lectures are the official slides and are the ones that should be used for revision.



Lecture 02A

More Advanced Java Topics

Horia A. Maior and Marjahan Begum

Topics for this Week

</>

- Lecture 02A:
 - OO and Java Refresher (2/2)
 - The missing bits from last lecture
 - Java Collections framework
 - Implementation of object oriented concepts in Java
- Lab 02:
 - Implementing the ZooApp
- Lecture 02B:
 - IDEs + Java Releases
 - jShell
 - Maintaining the ZooApp (basic maintenance)

Learning Outcomes for this Week

</>

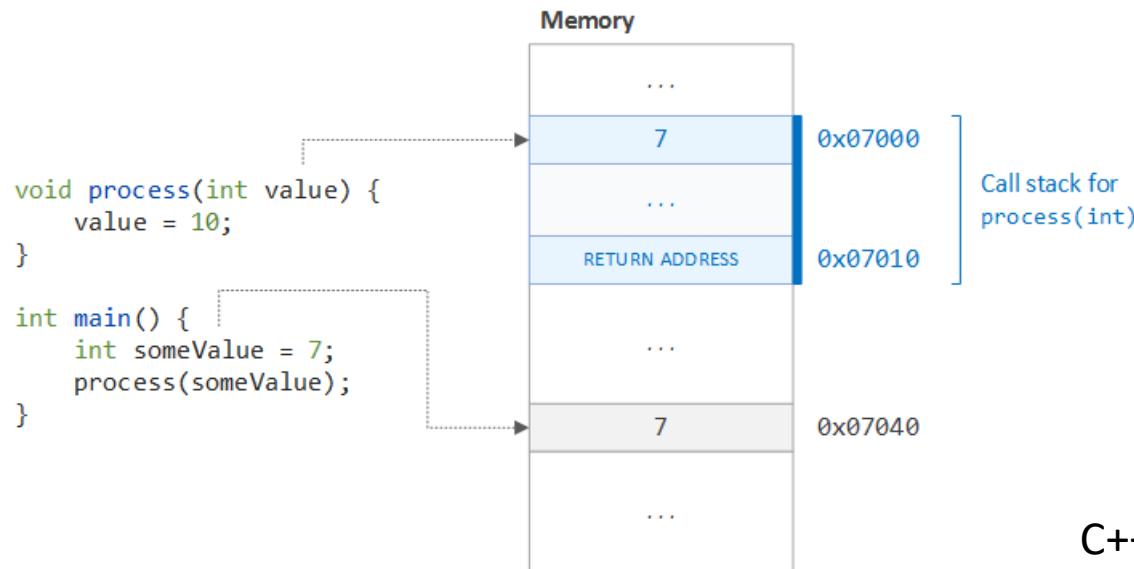
- At the end of this week ...
 - You should have a foundational knowledge of Java
 - You should be able to implement all types of relationships and concepts
 - You should have a first idea about approaching basic software maintenance

the missing bits from last lecture

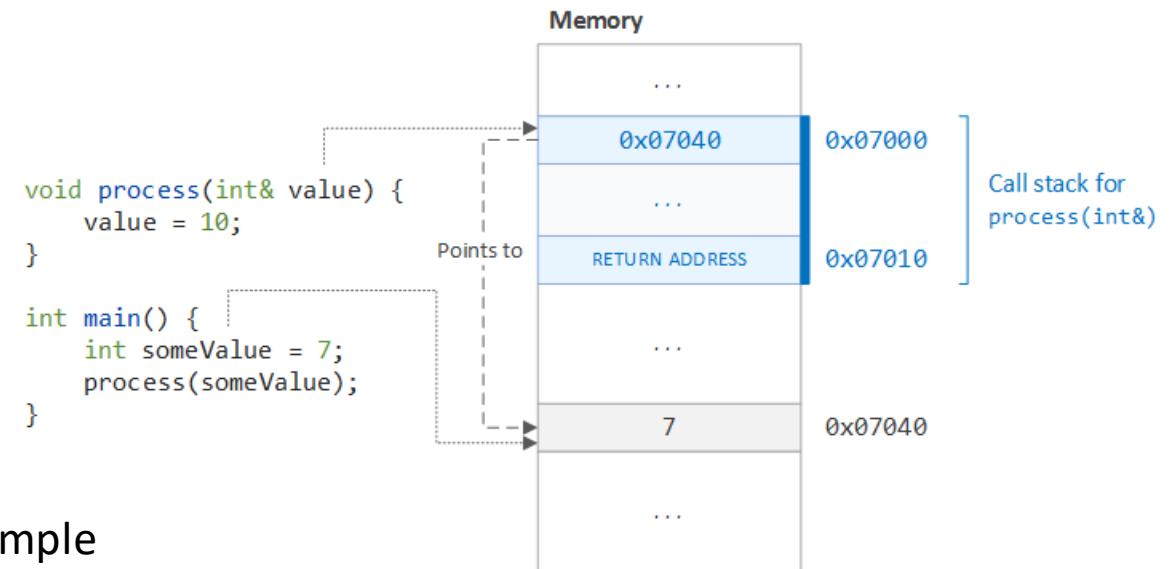
The Principle: Passing by Value vs Passing by Reference

</>

- Passing by value constitutes copying of data, where changes to the copied value are not reflected in the original value



- Passing by reference constitutes the aliasing of data, where changes to the aliased value are reflected in the original value



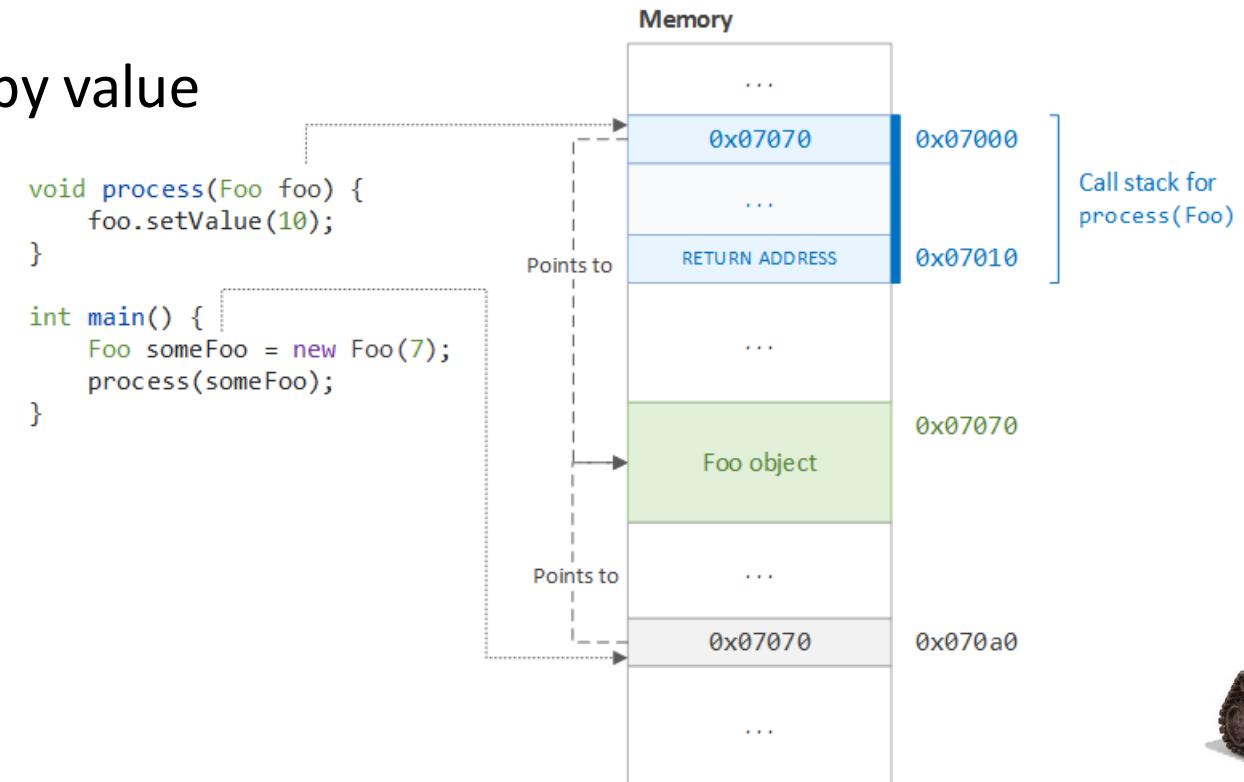
C++ Example

For more details see: <https://dzone.com/articles/pass-by-value-vs-reference-in-java>

Passing Data in Java: Always by Value!

</>

- The value associated with an object is actually a pointer, called an "object reference" to the object in memory
- Object references are passed by value



ZooProjectStage5

```
1 package com.Hagenbeck;
2
3 public class ZooApp {
4
5     public static void main(String[] args) {
6         int avgVisitors=100;
7
8         Zoo zool; // creating reference
9         zool=new Zoo( location: "Hamburg");
10        zool=new Zoo( location: "Munic"); // new object takes over reference; old object not accessible any more
11        Zoo zool2=zool; // references zool and zool2 point to same object
12        Zoo zool3=new Zoo();
13        System.out.println("\nZool:"+zool);
14        System.out.println("Zool2:"+zool2);
15        System.out.println("Zool3:"+zool3);
16        zool3.setLocation("Berlin");
17        zool.setLocation("Berlin"); // sets it for zool and zool2
18        System.out.println("\nZool:"+zool);
19        System.out.println("Zool2:"+zool2);
20        System.out.println("Zool3:"+zool3);
21        zool=new Zoo( location: "SanDiego"); // sets it for zool only
22        System.out.println("\nZool:"+zool);
23        System.out.println("Zool2:"+zool2);
24        System.out.println("Zool3:"+zool3);
25
26        zool.changeZoo(zool, avgVisitors); // passing object reference and primitive
27        System.out.println("\n"+zool+": "+avgVisitors);
28    }
29 }
30
31
```

Yankee Doodle Dandy (1942) - Theatrical Release Date: December 25, 1942

Project

Project ▾ ZooApp.java × Zoo.java × Manager.java

The screenshot shows the IntelliJ IDEA interface with the project tree on the left and the code editor on the right.

Project Tree:

- ZooProjectStage5 E:\Teaching\COMP201
- .idea
- out
- src
 - com.Hagenbeck
 - Manager
 - Zoo
 - ZooApp
 - module-info.java
- ZooProjectStage5.iml

Code Editor (Zoo.java):

```
1 package com.Hagenbeck;
2
3 public class Zoo {
4
5     private String location;
6
7     public Zoo() { this(location: "Unknown"); }
8
9
10    public Zoo(String location) {
11        //this.location=location;
12        this.setLocation(location);
13        System.out.println("\nConstructing "+this);
14    }
15
16
17    public void doSomething() { Manager manager=n;
18
19
20
21    public String getLocation() { return location;
22
23
24
25    public void setLocation(String location) { th
26
27    public void changeZoo(Zoo zoo, int avgVisitor
28        avgVisitors=200;
29        System.out.println("\n@Local Start:"+zoo);
30        zoo=new Zoo(location: "London");
31        //zoo.setLocation("Munich");
32        //this.setLocation("Amsterdam");
33        System.out.println("\n@Local End:"+zoo);
34    }
35
36
37    @Override
38    public String toString() { return getClass().n
39
40
41 }
```

A red rounded rectangle highlights the following code block in the `changeZoo` method:

```
avgVisitors=200;
System.out.println("\n@Local Start:"+zoo);
zoo=new Zoo(location: "London");
//zoo.setLocation("Munich");
//this.setLocation("Amsterdam");
System.out.println("\n@Local End:"+zoo);
```

4 1 1 1 1

Structure

Favorites

Todo Problems Build Terminal

1 Event Log





File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage5 - Manager.java

- □ X

ZooProjectStage5 > src > com > Hagenbeck > Manager



ZooApp



Project

Project + - ☰ ZooApp.java × c Zoo.java × c Manager.java ×



ZooProjectStage5 E:\Teaching\COMP20
> .idea
> out
src
 com.Hagenbeck
 Manager
 Zoo
 ZooApp
 module-info.java
 ZooProjectStage5.iml
> External Libraries
> Scratches and Consoles

1 package com.Hagenbeck;

2

3 public class Manager {

4 }

5 |

Structure

Favorites

TODO Problems Build Terminal

1 Event Log

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

CUIVIPZU13 </>

5:1 CRLF UTF-8 4 spaces ↴



Run: ZooApp

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\lib\idea_rt.jar=50647:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\bin" -Dfile.encoding=UTF-8
```

Constructing Zoo(Hamburg)

Constructing Zoo(Munic)

Constructing Zoo(Unknown)

Zoo1:Zoo(Munic)

Zoo2:Zoo(Munic)

Zoo3:Zoo(Unknown)

Zoo1:Zoo(Berlin)

Zoo2:Zoo(Berlin)

Zoo3:Zoo(Berlin)

Constructing Zoo(SanDiego)

Zoo1:Zoo(SanDiego)

Zoo2:Zoo(Berlin)

Zoo3:Zoo(Berlin)

@Local Start:Zoo(SanDiego)

Constructing Zoo(London)

@Local End:Zoo(London)

Zoo(SanDiego):100

Process finished with exit code 0

Structure

Favorites

- Using "this" inside methods
 - Inside a method, "this" refers to the object for which the method was called. "this" can be passed to other constructors and methods as a parameter.
- Return statements in void methods
 - A void method can use a return statement to quit the method early
 - There is no need for a return at the end



ZooProjectStage6

- Lets Create and add:
- A "Manager" object (and passing "this" as parameter) in "doSomething" method in "Zoo" class and
- adding "sayHello" method call;
- "doSomething" method call to "Zoo" class constructor;
- constructor to "Manager" class;
- add "sayHello" method with "return" statement (to stop method execution) to "Manager" class



ZooProjectStage6

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage6 - ZooApp.java

ZooProjectStage6 > src > com > Hagenbeck > ZooApp

Project ZooProjectStage6 E:\Teaching\COMP2013-G52DMS\G52D .idea out src com.Hagenbeck Manager Zoo ZooApp module-info.java ZooProjectStage6.iml External Libraries Scratches and Consoles

Structure Favorites

ZooApp.java X Zoo.java X Manager.java X

```
1 package com.Hagenbeck;
2
3 public class ZooApp {
4
5     public static void main(String[] args) {
6         int avgVisitors=100;
7
8         Zoo zool; // creating reference
9         zool=new Zoo( location: "Hamburg");
10        zool=new Zoo( location: "Munich"); // new object takes over reference; old object not accessible any more
11        Zoo zool2=zool; // references zool and zool2 point to same object
12        Zoo zool3=new Zoo();
13        System.out.println("\nZool:"+zool);
14        System.out.println("Zool2:"+zool2);
15        System.out.println("Zool3:"+zool3);
16        zool3.setLocation("Berlin");
17        zool.setLocation("Berlin"); // sets it for zool and zool2
18        System.out.println("\nZool:"+zool);
19        System.out.println("Zool2:"+zool2);
20        System.out.println("Zool3:"+zool3);
21        zool=new Zoo( location: "SanDiego"); // sets it for zool only
22        System.out.println("\nZool:"+zool);
23        System.out.println("Zool2:"+zool2);
24        System.out.println("Zool3:"+zool3);
25
26        zool.changeZoo(zool, avgVisitors); // passing object reference and primitive
27        System.out.println("\n"+zool+": "+avgVisitors);
28    }
29
30
31 }
```

Event Log 1 Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (moments ago) CUIVIPZU13 </>

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage6 - Zoo.java

ZooProjectStage6 > src > com > Hagenbeck > c Zoo

Project ZooApp.java Zoo.java Manager.java

1 package com.Hagenbeck;
2
3 public class Zoo {
4
5 private String location;
6
7 public Zoo() { this("Unknown"); }
8
9 public Zoo(String location) {
10 //this.location=location;
11 this.setLocation(location);
12 System.out.println("\nConstructing "+this);
13 doSomething();
14 }
15
16 public void doSomething() {
17 Manager manager=new Manager(this);
18 manager.sayHello();
19 }
20
21 public String getLocation() { return location; }
22
23 public void setLocation(String location) { this.location = location; }
24 public void changeZoo(Zoo zoo, int avgVisitors) {
25 avgVisitors=200;
26 System.out.println("\n@Local Start:"+zoo);
27 zoo=new Zoo(location: "London");
28 //zoo.setLocation("Munic");
29 //this.setLocation("Amsterdam");
30 System.out.println("\n@Local End:"+zoo);
31 }
32
33 @Override
34 public String toString() { return getClass().getSimpleName()+"("+this.getLocation()+")"; }
35
36 }

Scratches and Consoles

External Libraries

Structure

Favorites

TODO Problems Build Terminal Event Log

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

CUVIPZU13 </>

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage6 - Manager.java

ZooProjectStage6 > src > com > Hagenbeck > Manager

Project ZooProjectStage6 E:\Teaching\COMP2013-G52DMS\G52D .idea out src com.Hagenbeck Manager Zoo ZooApp module-info.java ZooProjectStage6.iml External Libraries Scratches and Consoles

Structure Favorites

ZooApp.java > Zoo.java > Manager.java

Manager.java

```
1 package com.Hagenbeck;
2
3 public class Manager {
4
5     private Zoo zoo;
6
7     public Manager(Zoo zoo) {
8         this.setZoo(zoo);
9         System.out.println("Constructing "+this);
10    }
11
12     public void sayHello() {
13         //boolean returnEarly=false;
14         boolean returnEarly=true;
15         if(returnEarly) {
16             System.out.println(this+":returnEarly");
17             return;
18         }
19         System.out.println(this+":returnLate");
20     }
21
22     public Zoo getZoo() { return zoo; }
23
24     public void setZoo(Zoo zoo) { this.zoo = zoo; }
25
26     public String toString() { return getClass().getSimpleName()+"("+zoo.getLocation()+")"; }
27
28 }
29
30
31
32
33
34
```

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

CUVIPZU13 </>

Event Log 1

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\lib\idea_rt.jar=50670:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\bin" -Dfile.encoding=UTF-8
```

```
Constructing Zoo(Hamburg)
Constructing Manager(Hamburg)
Manager(Hamburg):returnEarly
```

```
Constructing Zoo(Munic)
Constructing Manager(Munic)
Manager(Munic):returnEarly
```

```
Constructing Zoo(Unknown)
Constructing Manager(Unknown)
Manager(Unknown):returnEarly
```

```
Zoo1:Zoo(Munic)
Zoo2:Zoo(Munic)
Zoo3:Zoo(Unknown)
```

```
Zoo1:Zoo(Berlin)
Zoo2:Zoo(Berlin)
Zoo3:Zoo(Berlin)
```

```
Constructing Zoo(SanDiego)
Constructing Manager(SanDiego)
Manager(SanDiego):returnEarly
```

```
Zoo1:Zoo(SanDiego)
Zoo2:Zoo(Berlin)
Zoo3:Zoo(Berlin)
```

```
@Local Start:Zoo(SanDiego)
```

```
Constructing Zoo(London)
Constructing Manager(London)
Manager(London):returnEarly
```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage6 - Manager.java

ZooProjectStage6 > src > com > Hagenbeck > Manager

Project Run: ZooApp Manager.java

```
Manager(Munic):returnEarly  
Constructing Zoo(Unknown)  
Constructing Manager(Unknown)  
Manager(Unknown):returnEarly  
  
Zoo1:Zoo(Munic)  
Zoo2:Zoo(Munic)  
Zoo3:Zoo(Unknown)  
  
Zoo1:Zoo(Berlin)  
Zoo2:Zoo(Berlin)  
Zoo3:Zoo(Berlin)  
  
Constructing Zoo(SanDiego)  
Constructing Manager(SanDiego)  
Manager(SanDiego):returnEarly  
  
Zoo1:Zoo(SanDiego)  
Zoo2:Zoo(Berlin)  
Zoo3:Zoo(Berlin)  
  
@Local Start:Zoo(SanDiego)  
  
Constructing Zoo(London)  
Constructing Manager(London)  
Manager(London):returnEarly  
  
@Local End:Zoo(London)  
  
Zoo(SanDiego):100  
  
Process finished with exit code 0
```

Run TODO Problems Build Terminal

All files are up-to-date (a minute ago)

UK | CHINA | MALAYSIA

ZOOAPP2013 </>

34:1 CRLF UTF-8 Tab* 19

- Overloaded Methods
 - Methods of the **same class** that have the **same name** but **different numbers or types of parameters**
 - The compiler treats overloaded methods as completely different methods
 - Return Type Doesn't Matter
 - Compile-Time Resolution. This is known as compile-time polymorphism or static polymorphism.
 - The compiler knows which one to call based on the number and the types of the parameters passed to the method
 - The return type alone is not sufficient for making a distinction between overloaded methods

- A static field (class field or class variable) is shared by all objects of the class
- A non-static field (instance field or instance variable) belongs to an individual object
- Static fields are stored with the class code, separately from instance variables that describe an individual object
- **Modifying a Static Field Affects All Instances**

- Public static fields, usually global constants, are referred to in other classes using dot notation
 - `ClassName.constName`
- Usually static fields are **NOT** initialised in constructors; they are initialised either in declarations or in public static methods or just use their default value
- If a class has only static fields, there is no point in creating objects of that class, as all of them would be identical
 - Math and System are examples of the above (they have no public constructors and cannot be instantiated)

- Static methods can access and manipulate class's static fields. They belong to the class - not an instance of it.
- Static methods are called using dot notation
 - `ClassName.statMethod(...)`
- Note that **static methods cannot access instance fields or call instance methods** of the class while **instance methods can access all fields and call all methods** of their class - both static and non-static
- Static methods will usually take input from the parameters, perform actions on it, then return some result.

Static Methods

</>

- When to use static methods (Stack Overflow)
 - <https://stackoverflow.com/questions/2671496/when-to-use-static-methods>
- Stage 7 (more coding):
 - Adding static field "numZoosCreated" to "Zoo" class;
 - increasing "numZoosCreated" in Zoo class constructor
 - adding "numZoosCreated" getter to Zoo class;
 - Check "numZoosCreated" in "ZooApp" class



ZooProjectStage7

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage7 - ZooApp.java

ZooProjectStage7 > src > com > Hagenbeck > ZooApp

Project

ZooProjectStage7 E:\Teaching\COMP2013-G52DMS\G52D
|.idea
out
src
com.Hagenbeck
Manager
Zoo
ZooApp
module-info.java
ZooProjectStage7.iml
External Libraries
Scratches and Consoles

Structure

Favorites

ZooApp.java Manager.java Zoo.java

```
1 package com.Hagenbeck;
2
3 public class ZooApp {
4
5     public static void main(String[] args) {
6         int avgVisitors=100;
7
8         Zoo zoo1; // creating reference
9         zoo1=new Zoo( location: "Hamburg");
10        zoo1=new Zoo( location: "Munich"); // new object takes over reference; old object not accessible any more
11        Zoo zoo2=zoo1; // references zoo1 and zoo2 point to same object
12        Zoo zoo3=new Zoo();
13        System.out.println("\nzoo1:"+zoo1);
14        System.out.println("Zoo2:"+zoo2);
15        System.out.println("Zoo3:"+zoo3);
16        zoo3.setLocation("Berlin");
17        zoo1.setLocation("Berlin"); // sets it for zoo1 and zoo2
18        System.out.println("\nzoo1:"+zoo1);
19        System.out.println("Zoo2:"+zoo2);
20        System.out.println("Zoo3:"+zoo3);
21        zoo1=new Zoo( location: "SanDiego"); // sets it for zoo1 only
22        System.out.println("\nzoo1:"+zoo1);
23        System.out.println("Zoo2:"+zoo2);
24        System.out.println("Zoo3:"+zoo3);
25
26        zoo1.changeZoo(zoo1, avgVisitors); // passing object reference and primitive
27        System.out.println("\n"+zoo1":"+avgVisitors);
28        System.out.println("\ngetNumZoosCreated(): "+Zoo.getNumZoosCreated());
29    }
30
31
32 }
```

TODO Problems Build Terminal

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't sh... (moments ago) Checking for JDK updates 31:1 CRLF UTF-8 Tab* Event Log

UK | CHINA | MALAYSIA

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage7 - Zoo.java

ZooProjectStage7 > src > com > Hagenbeck > c Zoo

Project

ZooProjectStage7 E:\Teaching\COMP2013-G52DMS\G52D
|.idea
out
src
com.Hagenbeck
Manager
Zoo
ZooApp
module-info.java
ZooProjectStage7.iml

External Libraries
Scratches and Consoles

ZooApp.java Manager.java Zoo.java

```
public class Zoo {  
    private static int numZoosCreated;  
    private String location;  
  
    public Zoo() { this(location: "Unknown"); }  
  
    public Zoo(String location) {  
        numZoosCreated++;  
        //this.location=location;  
        this.setLocation(location);  
        System.out.println("\nConstructing "+this+" No:"+numZoosCreated);  
        doSomething();  
    }  
  
    public void doSomething() {  
        Manager manager=new Manager(zoo: this);  
        manager.sayHello();  
    }  
  
    public String getLocation() { return location; }  
  
    public void setLocation(String location) { this.location = location; }  
    public void changeZoo(Zoo zoo, int avgVisitors) {  
        avgVisitors=200;  
        System.out.println("\n@Local Start:"+zoo);  
        zoo=new Zoo(location: "London");  
        //zoo.setLocation("Munic");  
        //this.setLocation("Amsterdam");  
        System.out.println("\n@Local End:"+zoo);  
    }  
  
    public static int getNumZoosCreated() { return numZoosCreated; }  
    @Override  
    public String toString() { return getClass().getSimpleName()+"("+this.getLocation()+")"; }  
}
```

Structure

Favorites

TODO Problems Build Terminal Event Log

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

CUVIPZU13 </>

3:14 CRLF UTF-8 Tab* ↻

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage7 - Manager.java

ZooProjectStage7 > src > com > Hagenbeck > Manager

Project ZooProjectStage7 E:\Teaching\COMP2013-G52DMS\G52D .idea out src com.Hagenbeck Manager Zoo ZooApp module-info.java ZooProjectStage7.iml External Libraries Scratches and Consoles

Structure Favorites

ZooApp.java Manager.java Zoo.java

```
1 package com.Hagenbeck;
2
3 public class Manager {
4
5     private Zoo zoo;
6
7     public Manager(Zoo zoo) {
8         this.setZoo(zoo);
9         System.out.println("Constructing "+this);
10    }
11
12    public void sayHello() {
13        //boolean returnEarly=false;
14        boolean returnEarly=true;
15        if(returnEarly) {
16            System.out.println(this+":returnEarly");
17            return;
18        }
19        System.out.println(this+":returnLate");
20    }
21
22    public Zoo getZoo() { return zoo; }
23
24    public void setZoo(Zoo zoo) { this.zoo = zoo; }
25
26    public String toString() { return getClass().getSimpleName()+"("+zoo.getLocation()+")"; }
27
28}
29
30
31
32
33
34
```

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

CUVIPZU13 </>

Event Log 34:1 CRLF UTF-8 Tab* /

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\lib\idea_rt.jar=50686:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.2\bin" -Dfile.encoding=UTF-8
```

Constructing Zoo(Hamburg) No:1
Constructing Manager(Hamburg)
Manager(Hamburg):returnEarly

Constructing Zoo(Munic) No:2
Constructing Manager(Munic)
Manager(Munic):returnEarly

Constructing Zoo(Unknown) No:3
Constructing Manager(Unknown)
Manager(Unknown):returnEarly

Zoo1:Zoo(Munic)
Zoo2:Zoo(Munic)
Zoo3:Zoo(Unknown)

Zoo1:Zoo(Berlin)
Zoo2:Zoo(Berlin)
Zoo3:Zoo(Berlin)

Constructing Zoo(SanDiego) No:4
Constructing Manager(SanDiego)
Manager(SanDiego):returnEarly

Zoo1:Zoo(SanDiego)
Zoo2:Zoo(Berlin)
Zoo3:Zoo(Berlin)

@Local Start:Zoo(SanDiego)

Constructing Zoo(London) No:5
Constructing Manager(London)
Manager(London):returnEarly

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ZooProjectStage7 - Manager.java

ZooProjectStage7 > src > com > Hagenbeck > Manager

Project Run: ZooApp

Constructing Zoo(Unknown) No:3
Constructing Manager(Unknown)
Manager(Unknown):returnEarly

Zoo1:Zoo(Munic)
Zoo2:Zoo(Munic)
Zoo3:Zoo(Unknown)

Zoo1:Zoo(Berlin)
Zoo2:Zoo(Berlin)
Zoo3:Zoo(Berlin)

Constructing Zoo(SanDiego) No:4
Constructing Manager(SanDiego)
Manager(SanDiego):returnEarly

Zoo1:Zoo(SanDiego)
Zoo2:Zoo(Berlin)
Zoo3:Zoo(Berlin)

@Local Start:Zoo(SanDiego)

Constructing Zoo(London) No:5
Constructing Manager(London)
Manager(London):returnEarly

@Local End:Zoo(London)

Zoo(SanDiego):100

getNumZoosCreated(): 5

Process finished with exit code 0

Run TODO Problems Build Terminal

All files are up-to-date (a minute ago)

UK | CHINA | MALAYSIA

CUIMIZU13 </>

34:1 CRLF UTF-8 Tab* 29 Event Log

Static Fields and Methods

</>

- Does this compile?

```
public static void main(String[] args) {  
    int avgVisitors=100;  
  
    Zoo zoo1; // creating reference  
    zoo1=new Zoo( location: "Hamburg");  
    zoo1=new Zoo( location: "Munic"); // new object takes over reference; old object not accessible any more  
    Zoo zoo2=zoo1; // references zoo1 and zoo2 point to same object  
    Zoo zoo3=new Zoo();  
    System.out.println("\nzoo1:"+zoo1);  
    System.out.println("Zoo2:"+zoo2);  
    System.out.println("Zoo3:"+zoo3);  
    zoo3.setLocation("Berlin");  
    zoo1.setLocation("Berlin"); // sets it for zoo1 and zoo2  
    System.out.println("\nzoo1:"+zoo1);  
    System.out.println("Zoo2:"+zoo2);  
    System.out.println("Zoo3:"+zoo3);  
    zoo1=new Zoo( location: "SanDiego"); // sets it for zoo1 only  
    System.out.println("\nzoo1:"+zoo1);  
    System.out.println("Zoo2:"+zoo2);  
    System.out.println("Zoo3:"+zoo3);  
  
    zoo1.changeZoo(zoo1, avgVisitors); // passing object reference and primitive  
    System.out.println("\n"+zoo1+":"+avgVisitors);  
    System.out.println("\ngetNumZoosCreated(): "+Zoo.getNumZoosCreated());  
    test();  
}  
  
public void test() {  
    System.out.println("This is a Test");  
}
```



ZooProjectStage8

ZooProjectStage8 > src > com > Hagenbeck > ZooApp > main



Project

Project

- ZooProjectStage8 E:\Teaching\COMP2013-G52DMS\G52D
- .idea
- out
- src
 - com.Hagenbeck
 - Manager
 - Zoo
 - ZooApp
 - module-info.java
 - ZooProjectStage8.iml
- External Libraries
- Scratches and Consoles

ZooApp.java

```
8     Zoo zoo1; // creating reference
9     zoo1=new Zoo( location: "Hamburg");
10    zoo1=new Zoo( location: "Munic"); // new object takes over reference; old object not accessible any more
11    Zoo zoo2=zoo1; // references zoo1 and zoo2 point to same object
12    Zoo zoo3=new Zoo();
13    System.out.println("\nZoo1:"+zoo1);
14    System.out.println("Zoo2:"+zoo2);
15    System.out.println("Zoo3:"+zoo3);
16    zoo3.setLocation("Berlin");
17    zoo1.setLocation("Berlin"); // sets it for zoo1 and zoo2
18    System.out.println("\nZoo1:"+zoo1);
19    System.out.println("Zoo2:"+zoo2);
20    System.out.println("Zoo3:"+zoo3);
21    zoo1=new Zoo( location: "SanDiego"); // sets it for zoo1 only
22    System.out.println("\nZoo1:"+zoo1);
23    System.out.println("Zoo2:"+zoo2);
24    System.out.println("Zoo3:"+zoo3);
25
26    zoo1.changeZoo(zoo1, avgVisitors); // passing object reference and primitive
27    System.out.println("\n"+zoo1+":"+avgVisitors);
28    System.out.println("\ngetNumZoosCreated(): "+Zoo.getNumZoosCreated());
29    test();
```

Build: Build Output

- ZooProjectStage8: build failed At 08/10/2021 08:40 with 1 error 4 sec, 461 ms
- ZooApp.java src\com\Hagenbeck 1 error
 - non-static method test() cannot be referenced from a static context :29

E:\Teaching\COMP2013-G52DMS\G52DMS 2021-2022\01 @ The Challenges of DMS\Lecture01B\ZooProjectStages\ZooProjectStage8\src\com\Hagenbeck\ZooApp.java:29
java: non-static method test() cannot be referenced from a static context

Structure

Favorites

TODO Problems Build Terminal

Event Log

Non-static method 'test()' cannot be referenced from a static context

29:9 CRLF UTF-8 Tab*



java collections framework



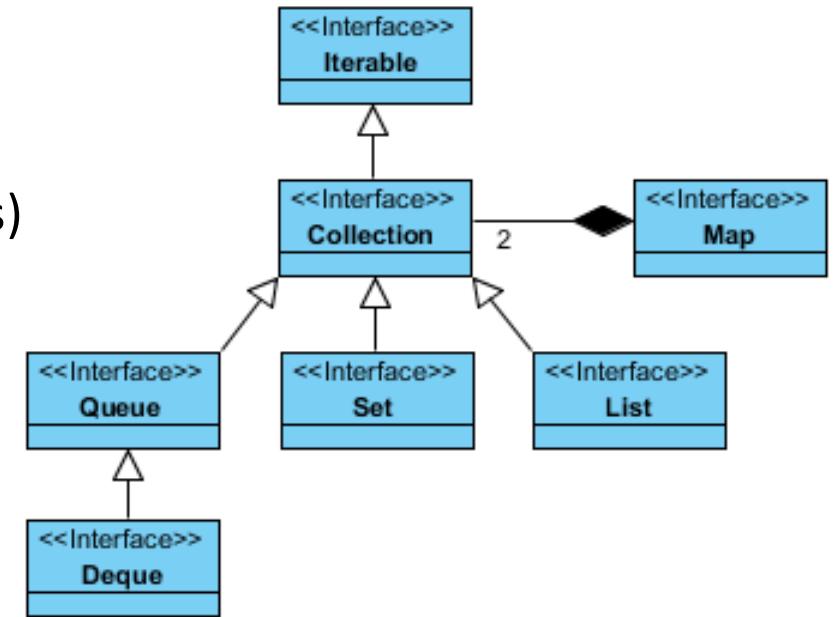
- What do we understand by "Collections" in Java?
 - A collection is an object that represents a group of objects
 - The Collections API is a unified framework for representing and manipulating collections, independent of their implementation
- What does the abbreviation API stand for?
 - Application Programming Interface
- What is the difference between a library and an API?
 - An API is an interface or communication protocol between a client and a server, intended to simplify the building of client-side software. A library contains re-usable chunks of code.

- Java Collections Framework (JCF) principle ideas:
 - We have container objects that contain objects
 - All containers are either "collections" or "maps"
 - All containers provide a common set of method signatures, in addition to their unique set of method signatures
- The Java Collections Framework contains data structures
 - e.g. arrays; lists; maps
- The Java Collections Framework contains algorithmic operations
 - e.g. searching; sorting

Java Collections Framework

</>

- Collection
 - Something that holds a dynamic collection of objects
- Map
 - Defines mapping between keys and objects (two collections)
- Iterable
 - Collections are able to return an iterator object that can scan over the contents of a collection one object at a time



- Core collection framework interfaces
 - **Iterable**: Represents an iterator object
 - **Collection**: Represents a group of objects (elements)
 - **Map**: Maps keys to values; no duplicate keys
 - **Queue**: Represents FIFO queues or LIFO stacks
 - **Deque**: Represents a double ended queue
 - **Set**: A collection that cannot contain duplicate elements
 - **List**: An ordered sequence of elements that allows duplicate elements
- Interface location
 - Most interfaces can be found in the `java.util.*` package
 - The "Iterable" interface resides in the `java.lang.*` package

Java Collections Framework

</>

- Classes that implement the collection interfaces typically have names in the form of <Implementation style><Interface>

Interface	Implementation style				
	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap	LinkedHashMap	LinkedHashMap

- Legacy classes (**do not use**)
 - Vector (now ArrayList); HashTable (now HashMap); Stack (now ArrayDeque)

For more information refer to: <https://nikhilmopidevi.github.io/2017/06/19/Overview-of-the-Java-Collections-Framework/>

Doubly-linked list and other data structures are well explained here: <https://www.javatpoint.com/doubly-linked-list>

Module java.base**Package** java.util

Class LinkedList<E>

```
java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.AbstractSequentialList<E>
        java.util.LinkedList<E>
```

Type Parameters:

E - the type of elements held in this collection

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, List<E>, Queue<E>

```
public class LinkedList<E>
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable
```

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

Note that this implementation is not synchronized. If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements; merely setting the value of an element is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the list. If no such object exists, the list should be "wrapped" using the Collections.synchronizedList method. This is best done at creation time, to prevent accidental unsynchronized access to the list:

```
List list = Collections.synchronizedList(new LinkedList(...));
```

The iterators returned by this class's iterator and listIterator methods are *fail-fast*: if the list is structurally modified at any time after the iterator is created, in any way except through the Iterator's own remove or add methods, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Constructor Summary

Constructors

Constructor

`LinkedList()`

"? extends E" means "some type that either is E or a subtype of E"

Constructs an empty list.

`LinkedList(Collection<? extends E> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method	Description
void	<code>add(int index, E element)</code>	Inserts the specified element at the specified position in this list.
boolean	<code>add(E e)</code>	Appends the specified element to the end of this list.
boolean	<code>addAll(int index, Collection<? extends E> c)</code>	Inserts all of the elements in the specified collection into this list, starting at the specified position.
boolean	<code>addAll(Collection<? extends E> c)</code>	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void	<code>addFirst(E e)</code>	Inserts the specified element at the beginning of this list.
void	<code>addLast(E e)</code>	Appends the specified element to the end of this list.
void	<code>clear()</code>	Removes all of the elements from this list.
Object	<code>clone()</code>	Returns a shallow copy of this <code>LinkedList</code> .

add

```
public void add(int index,  
                E element)
```

Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

Specified by:

[add in interface List<E>](#)

Overrides:

[add in class AbstractSequentialList<E>](#)

Parameters:

index - index at which the specified element is to be inserted

element - element to be inserted

Throws:

[IndexOutOfBoundsException](#) - if the index is out of range ($\text{index} < 0 \text{ || } \text{index} > \text{size}()$)

remove

```
public E remove(int index)
```

Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the list.

Specified by:

[remove in interface List<E>](#)

Overrides:

[remove in class AbstractSequentialList<E>](#)

Parameters:

- Non typesafe collections (**do not use**)
 - Collection constructors are not able to specify the type of objects the collection is intended to contain
 - Need to cast objects when using them
 - A "ClassCastException" will be thrown if we attempt to cast to the wrong type

The screenshot shows a Java code editor and a run window. The code in the editor is:

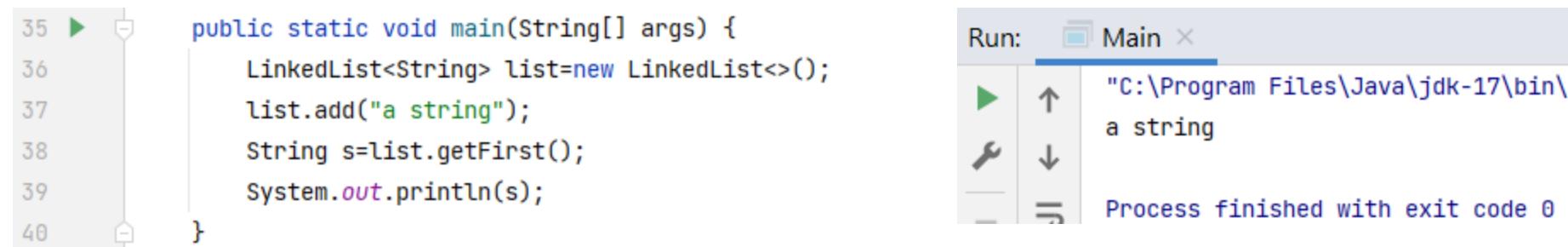
```
16 >  public static void main(String[] args) {  
17     LinkedList list=new LinkedList();  
18     list.add("a string");  
19     String s=(String)list.getFirst();  
20     System.out.println(s);  
21 }
```

The run window shows the output of the program:

Run: Main
C:\Program Files\Java\jdk-17\bin\
a string
Process finished with exit code 0



- Typesafe collections with "Generics"
 - Classes support generics by allowing a type variable to be included in their declaration; type are declared for the reference and constructor



The screenshot shows a Java code editor and a run window. The code in the editor is:

```
35 > public static void main(String[] args) {  
36     LinkedList<String> list=new LinkedList<>();  
37     list.add("a string");  
38     String s=list.getFirst();  
39     System.out.println(s);  
40 }
```

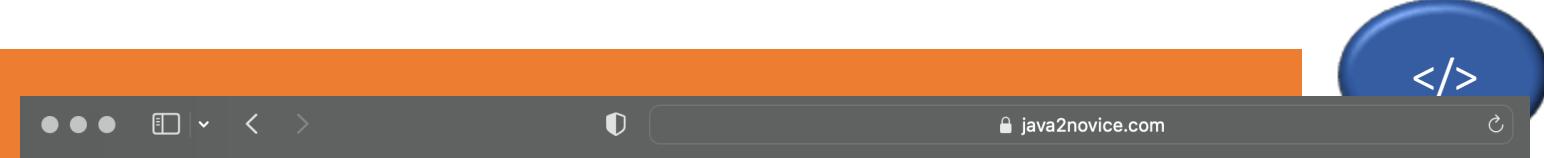
The run window shows the output of the program:

Run: Main
C:\Program Files\Java\jdk-17\bin\
a string
Process finished with exit code 0

- You cannot type a collection using a primitive type (e.g. int)
 - Values of primitive types need to be put into objects of a suitable wrapper class (e.g. Integer) before they can be added to a collection



Useful Resources



<http://www.java2novice.com/java-collections-and-util/>

Useful Resources

</>

Java Tutorial

- ➊ Java - Home
- ➋ Java - Overview
- ➌ Java - Environment Setup
- ➍ Java - Basic Syntax
- ➎ Java - Object & Classes
- ➏ Java - Basic Datatypes
- ➐ Java - Variable Types
- ➑ Java - Modifier Types
- ➒ Java - Basic Operators
- ➓ Java - Loop Control
- ➔ Java - Decision Making
- ➕ Java - Numbers
- ➖ Java - Characters
- ➗ Java - Strings
- ➘ Java - Arrays
- ➙ Java - Date & Time
- ➚ Java - Regular Expressions
- ➛ Java - Methods
- ➜ Java - Files and I/O
- ➝ Java - Exceptions
- ➞ Java - Inner classes

<https://www.tutorialspoint.com/java/>

Java Object Oriented

- ➊ Java - Inheritance
- ➋ Java - Overriding
- ➌ Java - Polymorphism
- ➍ Java - Abstraction
- ➎ Java - Encapsulation
- ➏ Java - Interfaces
- ➐ Java - Packages

Java Advanced

- ➊ Java - Data Structures
- ➋ Java - Collections
- ➌ Java - Generics
- ➍ Java - Serialization
- ➎ Java - Networking
- ➏ Java - Sending Email
- ➐ Java - Multithreading
- ➑ Java - Applet Basics
- ➒ Java - Documentation

implementation of object oriented concepts in java

Concepts we are looking at ...

</>

- Aggregation and Composition
- Inheritance and Polymorphism
- Abstract Methods and Classes
- Interfaces

Case Study: Zoo Management

</>

The screenshot shows the homepage of Chester Zoo's website. At the top, there is a navigation bar with links for "OUR ZOO", "ABOUT US", "ANIMALS" (which is underlined), "PLANTS & GARDENS", "ZOO MEMORIES", "SUPPORT US", "PLAN YOUR VISIT", "GIFTS & EXPERIENCES", "LATEST NEWS", "PREVENTING EXTINCTION", and a search icon. Below the navigation is a main banner featuring a lemur. The banner text includes "Meet our ANIMALS", "Discover the beautiful animals & habitats at Chester Zoo", and "BOOK YOUR TICKETS". At the bottom of the page, there is a purple footer bar with a "SEARCH" field, an "ANIMAL GROUP" dropdown menu, and a cookie consent message.

chesterzoo.org

CHESTERZOO

OUR ZOO ABOUT US ANIMALS PLANTS & GARDENS ZOO MEMORIES SUPPORT US

PLAN YOUR VISIT GIFTS & EXPERIENCES LATEST NEWS PREVENTING EXTINCTION

BOOK YOUR TICKETS

MORE ▾

Home > Our Zoo > Animals

Meet our
÷ANIMALS÷

Discover the beautiful animals & habitats at
Chester Zoo

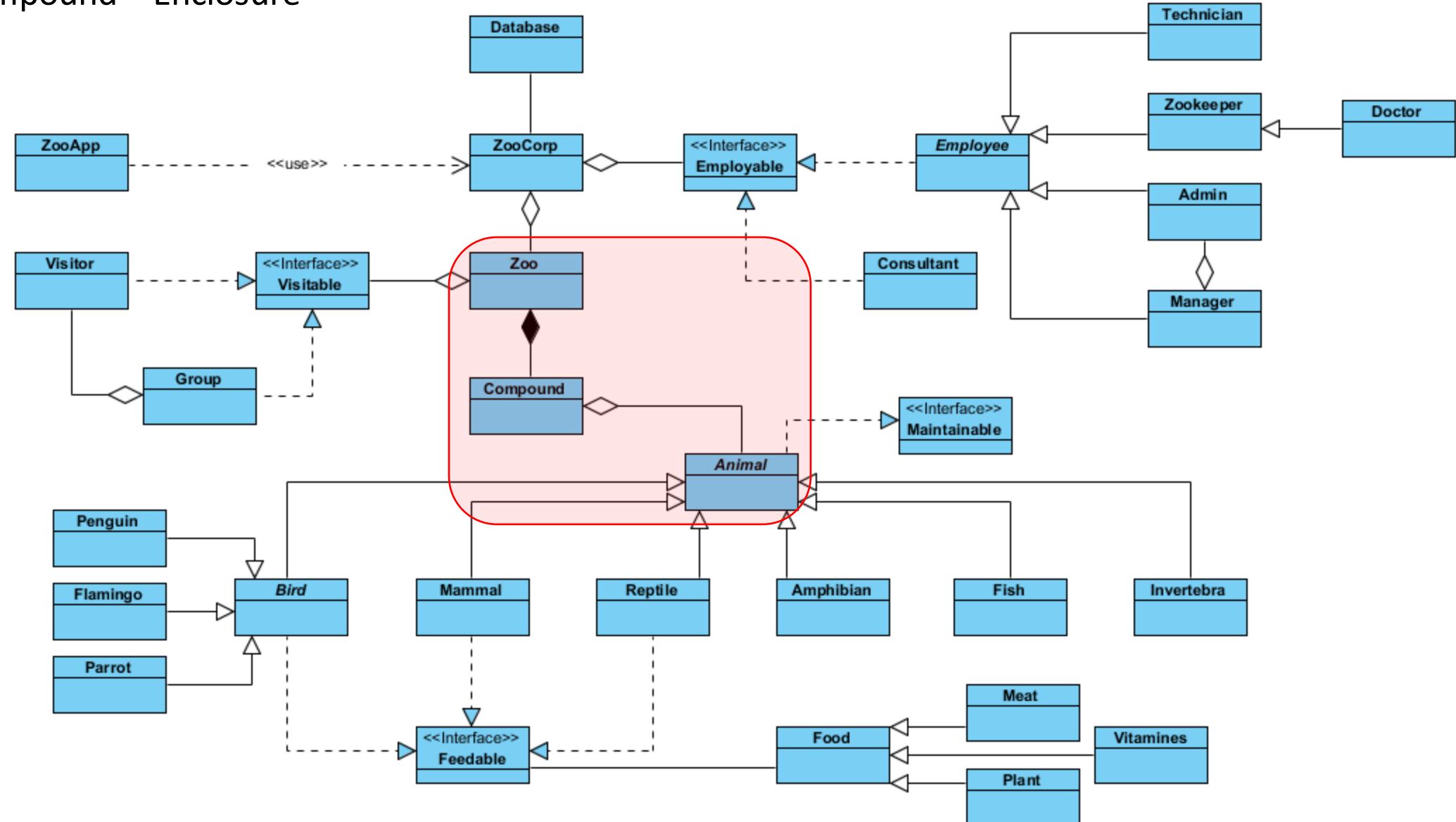
SEARCH

ANIMAL GROUP

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.

Reject All Accept All Cookies

NB: Compound = Enclosure



</>

Aggregation and Composition – 2 min discussion in pairs

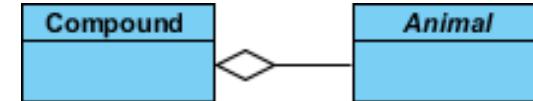
</>



- What is the difference between the Aggregations and Compositions?

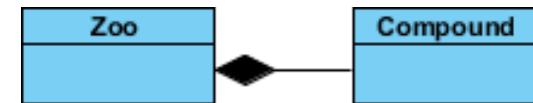
- Aggregation

- The object exists outside the other, is created outside, so it is passed as an argument



- Composition

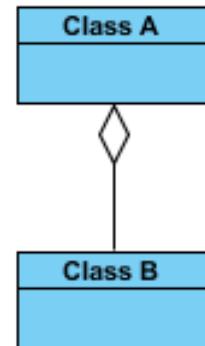
- The object only exists, or only makes sense inside the other, as a part of the other



Aggregation in Java

</>

- An object of class B **is part of** an object of class A (semantically) but the object of class B can be shared and if the object of class A is deleted, the object of class B is not deleted.
 - The object of class A stores the reference to the object of class B for later use
 - Often setter injection is used

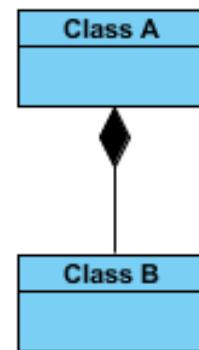


```
3 public class A {  
4     private B b;  
5  
6     public void setB(B b){  
7         this.b=b;  
8     }  
9 }
```

Composition in Java

</>

- An object of class A **owns** an object of class B and the object of class B cannot be shared and if the object of class A is deleted, the object of class B is also deleted
 - The containing object is responsible for the creation and life cycle of the contained object (either directly or indirectly)
 - Composition via member initialisation
 - Composition via constructor initialisation



```
3   public class A {  
4       private B b=new B();  
5   }
```

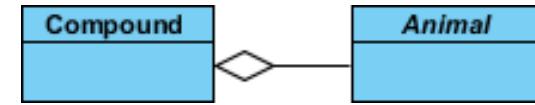
```
3   public class A {  
4       private B b;  
5  
6       public A(){  
7           this.b=new B();  
8       }  
9   }
```

Aggregation

</>



```
7  public class Compound {  
8      private ArrayList<Animal> animals;  
9  
10     public Compound() { animals=new ArrayList<>(); }  
11  
12     /*  
13         */  
14     public void addAnimal() {  
15         animals.add(new Animal());  
16     }  
17     */  
18     public void addAnimal(Animal animal) { animals.add(animal); }  
19  
20     public void printInfo() { System.out.println("The compound has "+animals.size()+" animals."); }  
21  
22 }  
23 }
```



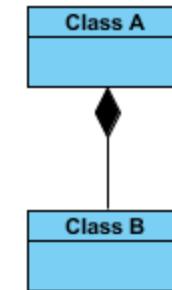
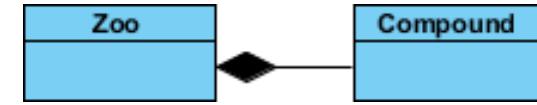
```
3  public abstract class Animal {  
4  }
```

Composition

```

5   public class Zoo {
6       private String location;
7       private ArrayList<Compound> compounds;
8
9       public Zoo(String location, int numCompounds) {
10           this.setLocation(location);
11           this.compounds=new ArrayList<Compound>();
12           createCompound(numCompounds);
13       }
14
15      public Zoo() { this(location: "Unknown", numCompounds: 1); }
16
17      public void createCompound(int numCompounds) {
18          if(numCompounds<1)numCompounds=1;
19          for(int i=0;i<numCompounds;i++) {
20              this.compounds.add(new Compound());
21          }
22      }
23
24      public String getLocation() { return location; }
25
26      public void setLocation(String location) { this.location = location; }
27
28      public void printInfo() { System.out.println("The zoo in "+location+" has "+compounds.size()+" compounds."); }
29
30
31
32
33
34
35
36
37

```



```

3  public class A {
4      private B b=new B();
5  }
6
7  public class A {
8      private B b;
9  }
10
11     public A(){
12         this.b=new B();
13     }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

- What is inheritance and why do we use it?
 - Inheritance: Forming new classes based on existing ones
 - Superclass: Parent class being extended
 - Subclass: Child class that inherits behaviour from the parent class
 - "Is-A" relationship

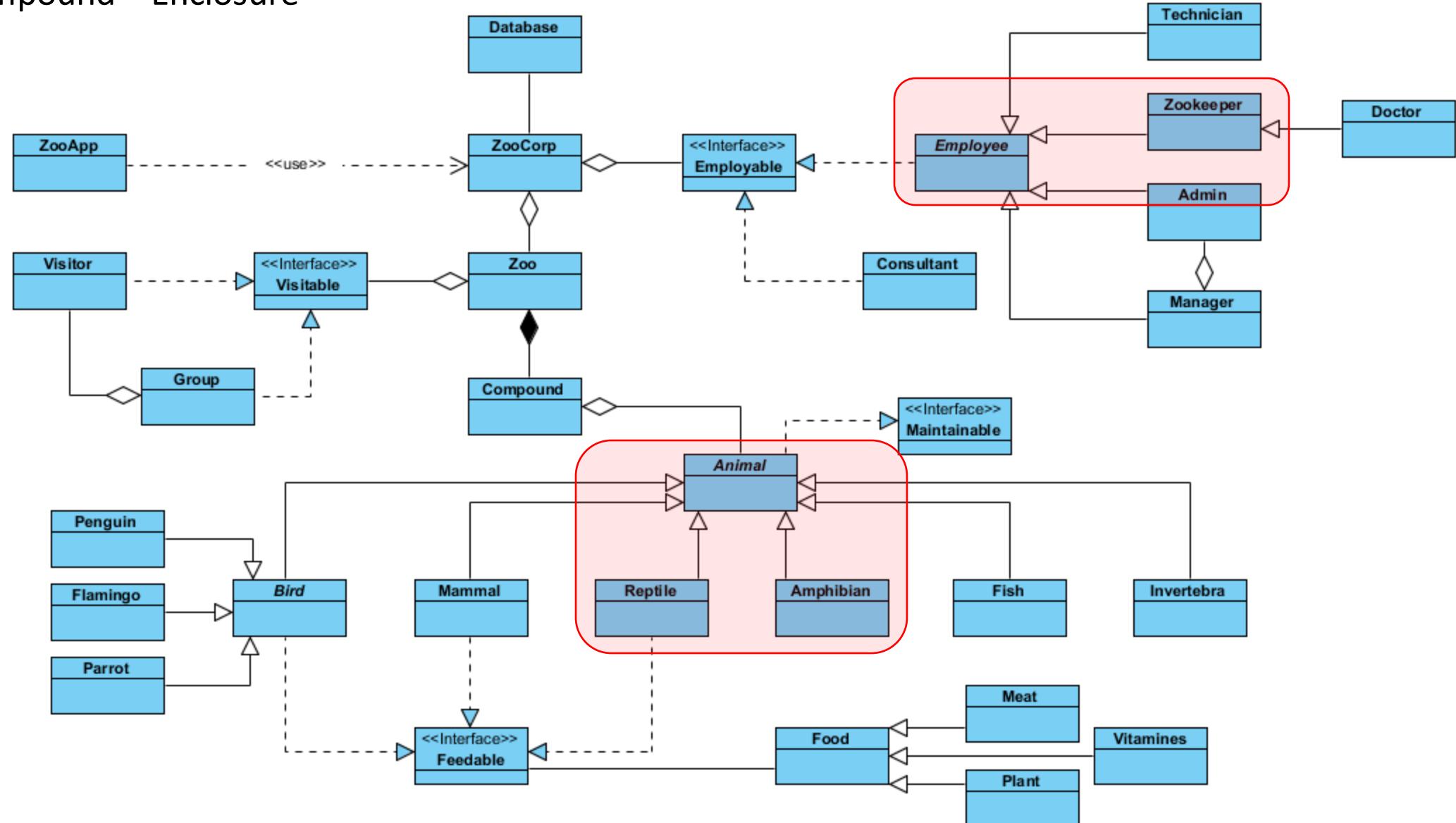
Polymorphism – Job Interview Question, 2 min, pairs

</>



- What is the difference between polymorphism, method overloading, and method overriding?
 - Polymorphism
 - Polymorphism is an object oriented concept
 - Method overloading and method overriding are two forms of polymorphism
 - Method overloading
 - Methods with same the name co-exists in the same class but they must have different method signature (number/type of parameters – think about constructors)
 - Resolved during compile time (static binding)
 - Method overriding
 - Method with the same name is declared in super and sub class (think about the bike1.currentState() example from labsheet 1)
 - Resolved during runtime

NB: Compound = Enclosure



</>

Inheritance & Polymorphism

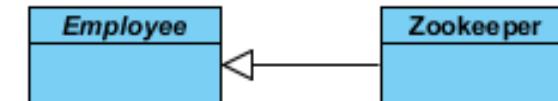
</>

```
3  public abstract class Employee {  
4      private String name;  
5      private double salary;  
6  
7      public Employee(String name) {  
8          setName(name);  
9          setSalary(2000);  
10     }  
11  
12     public String getName() { return name; }  
13  
14     public void setName(String name) { this.name = name; }  
15  
16     public double getSalary() { return salary; }  
17  
18     public void setSalary(double salary) { this.salary = salary; }  
19  
20     public abstract void promotion();  
21  
22 }  
23  
24  
25  
26  
27  
28  
29 }
```

```
3  public class Zookeeper extends Employee {  
4  
5      public Zookeeper(String name) { super(name); }  
6  
7      @Override  
8      public double getSalary() {  
9          double baseSalary=super.getSalary();  
10         return baseSalary+1000.00;  
11     }  
12  
13  
14  
15  
16     @Override  
17     public void promotion() { super.setSalary(super.getSalary()*1.1); }  
18  
19 }
```

– Notes:

- A subclass can call its parent's method(s) and constructor(s)
- A subclass can override its parent's method(s) but not its constructor(s)

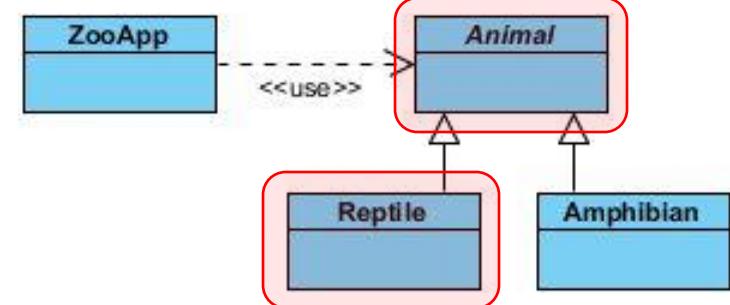


<https://stackoverflow.com/questions/5099924/is-constructor-overriding-possible>

Inheritance & Polymorphism

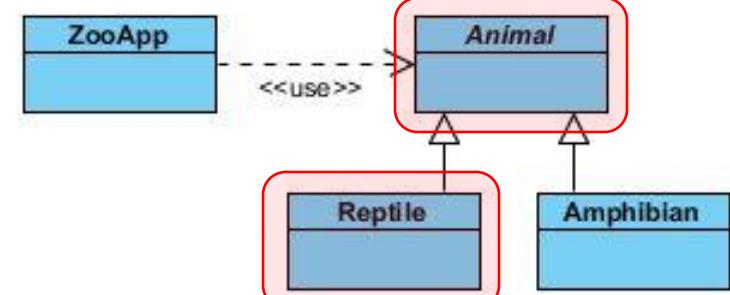
</>

```
3  o↓ public abstract class Animal {  
4      private String name;  
5  
6      public Animal(String name) { this.setName(name); }  
7  
8  
10  i↓     public abstract void eat();  
11  
12  o↓     public void enjoy() { System.out.println(this.getClass().getSimpleName()+" enjoys life as animal."); }  
13  
14  
15  
16      public String getName() { return name; }  
17  
18  
19  
20      public void setName(String name) { this.name = name; }  
21  
22 }  
23 }
```



Inheritance & Polymorphism

</>



```
3  public class Reptile extends Animal {  
4      private int numTeeth;  
5  
6      public Reptile(String name, int numTeeth) {  
7          super(name);  
8          this.setNumTeeth(numTeeth);  
9      }  
10  
11     @Override  
12     public void eat() { System.out.println(this.getClass().getSimpleName()+" eats like a reptile."); }  
13  
14     public int getNumTeeth() { return numTeeth; }  
15  
16     public void setNumTeeth(int numTeeth) { this.numTeeth = numTeeth; }  
17  
18 }  
19  
20  
21  
22  
23 }
```

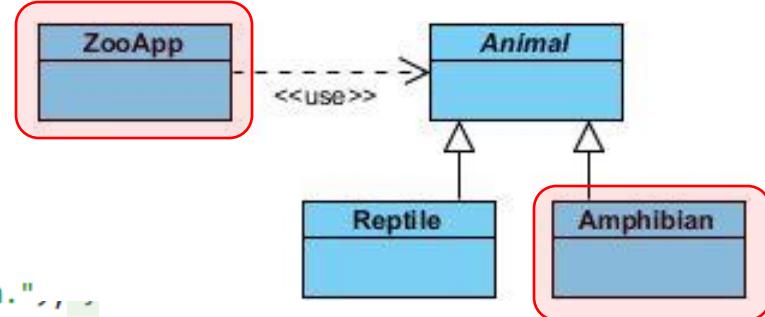
A code editor interface is shown, displaying Java code for a **Reptile** class. The code includes a constructor, an **@Override** annotated **eat()** method, and two accessors for a **numTeeth** field. Lines 12 and 13 are highlighted with a red box.



Inheritance & Polymorphism

</>

```
3  public class Amphibian extends Animal {  
4  
5      public Amphibian(String name) { super(name); }  
6  
7      @Override  
8      public void eat() { System.out.println(this.getClass().getSimpleName()+" eats like an amphibian."); }  
9  
10     @Override  
11     public void enjoy() { System.out.println(this.getClass().getSimpleName()+" enjoys life as amphibian."); }  
12  
13 }  
14  
15 }
```



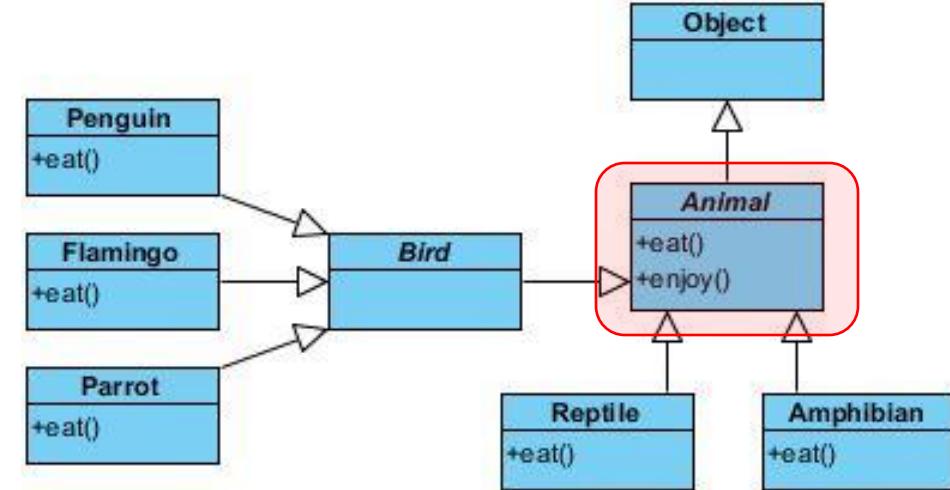
```
5 ►  public class ZooApp {  
6  
7 ►   public static void main(String[] args) {  
8     Animal animal1=new Amphibian( name: "Frog");  
9     Animal animal2=new Reptile( name: "Snake", numTeeth: 4);  
10    Reptile reptile=new Reptile( name: "Turtle", numTeeth: 24);  
11    animal1.enjoy();  
12    animal2.enjoy();  
13    reptile.enjoy();  
14  }  
15 }
```

```
Run: Main ×  
►  ⌂  ⌄  ⌅  ⌆  ⌇  ⌈  ⌉  ⌊  ⌋  
"C:\Program Files\Java\jdk-17\bin\java  
Amphibian enjoys life as amphibian.  
Reptile enjoys life as animal.  
Reptile enjoys life as animal.  
Process finished with exit code 0
```

Abstract Methods and Classes

</>

- Animal is an example of an abstract class
 - eat() is an abstract class
 - enjoy() is a concrete class
- Any subclass of class Animal has two choices:
 - Define an eat() method
 - Be abstract
- Keep in mind that abstract classes cannot be used to instantiate objects but references to abstract classes are legal (and encouraged)



Abstract Methods and Classes

</>

```
5 ► public class ZooApp {  
6  
7 ►     public static void main(String[] args) {  
8         ArrayList<Object> animals=new ArrayList<>();  
9         Object o=new Reptile( name: "Snake", numTeeth: 4);  
10        Reptile r=new Reptile( name: "Turtle", numTeeth: 24);  
11        animals.add(o);  
12        animals.add(r);  
13        animals.add(new Amphibian( name: "Frog"));  
14        while(animals.size()>0) {  
15            o=animals.remove( index: 0);  
16            System.out.println(o.toString());  
17            ((Animal)o).eat();  
18            ((Animal)o).enjoy();  
19            System.out.println();  
20        }  
21    }  
22 }
```

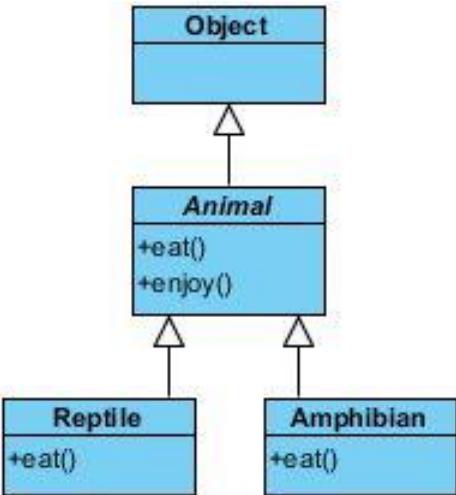
Run: Main ×

"C:\Program Files\Java\jdk-17\bin\j
com.zooproject.Reptile@5b6f7412
Reptile eats like a reptile.
Reptile enjoys life as animal.

com.zooproject.Reptile@3941a79c
Reptile eats like a reptile.
Reptile enjoys life as animal.

com.zooproject.Amphibian@506e1b77
Amphibian eats like an amphibian.
Amphibian enjoys life as amphibian.

Process finished with exit code 0



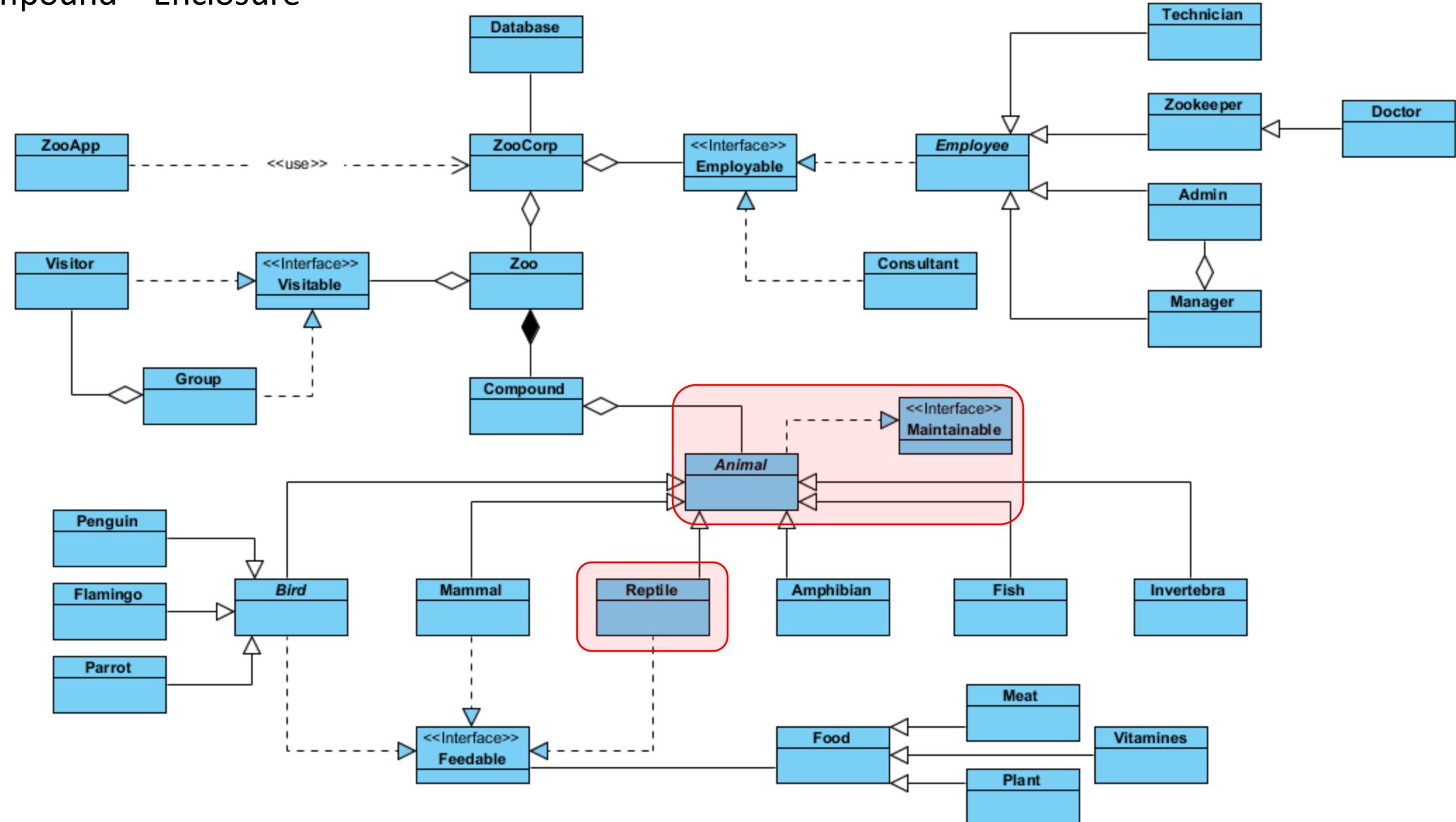
We need to cast the Object o into an Animal, in order to access methods eat() and enjoy() as they do not exist in the Object class

- An interface is an abstract type that is used to describe a behavior that classes must implement
- Interfaces may only contain method signature and constant declarations (variable declarations that are declared to be both static and final)
 - Starting with Java 8, default and public static methods can have an implementation in the interface definition
 - Starting with Java 9, private and private static methods can have an implementation in the interface definition

- Interfaces cannot be instantiated, but rather are implemented
- A class that implements an interface must implement all of the non-default methods described in the interface, or be an abstract class
- Interfaces are less restrictive when it comes to inheritance
 - While classes can only ever extend one other class (single inheritance), with interfaces we can choose to implement as many as we like
 - e.g. "public class A extends B implements C, D, E {...}"

- Some rules:
 - Use the keyword "interface" instead of "class" to declare an interface
 - Implement an interface with the "implements" keyword. A class that implements an interface must provide implementations for all the methods in the interface.
 - Generally all interfaces whose names end with "-able" mark the interface as being able to do something (e.g. Runnable - run(), Executable - execute(), Comparable - compareTo())
 - Similar to classes, you can build up inheritance hierarchies of interfaces by using the "extends" keyword

NB: Compound = Enclosure



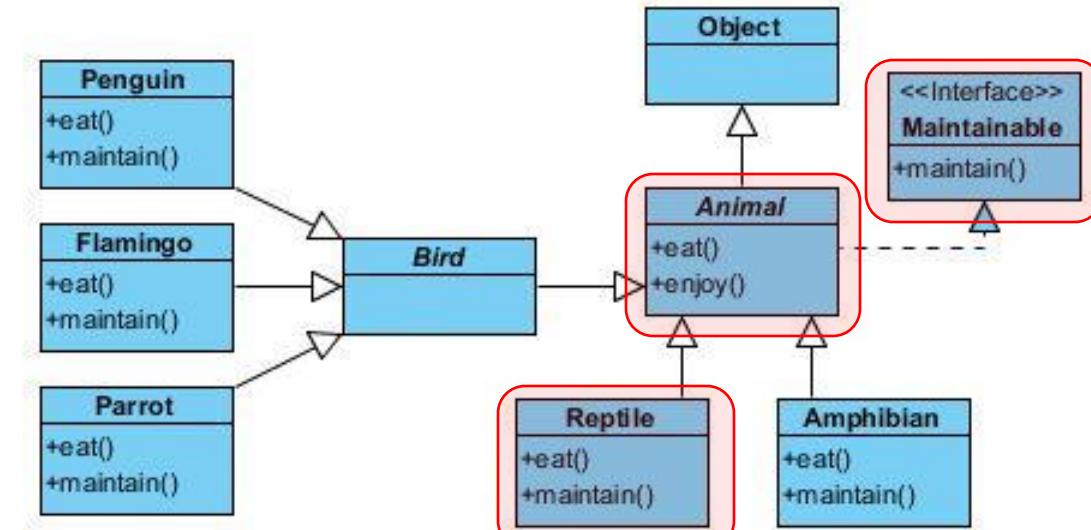
</>

Interfaces

</>

```
3  ↴ public interface Maintainable {  
4  
5  ↴     public void maintain();  
6 }
```

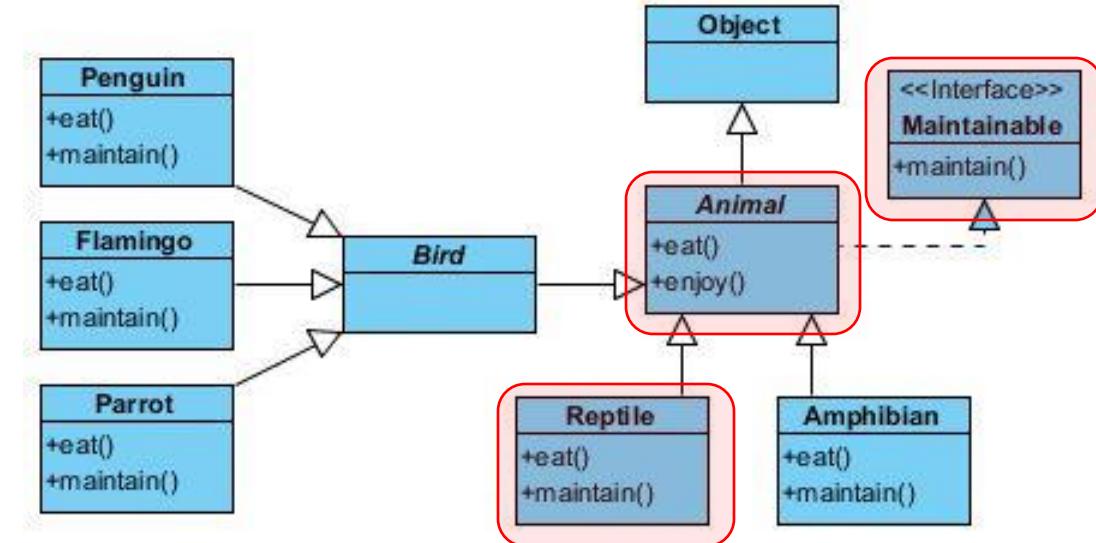
```
3 ⚪↓ public abstract class Animal implements Maintainable {  
4             private String name;  
5  
6             public Animal(String name) { this.setName(name); }  
7  
10 ↴     public abstract void eat();  
11  
12 ⚪↓     public void enjoy() { System.out.println(this.getClass().getSimpleName()+" enjoys life as animal."); }  
13  
16             public String getName() { return name; }  
17  
20             public void setName(String name) { this.name = name; }  
21 }
```



Interfaces

</>

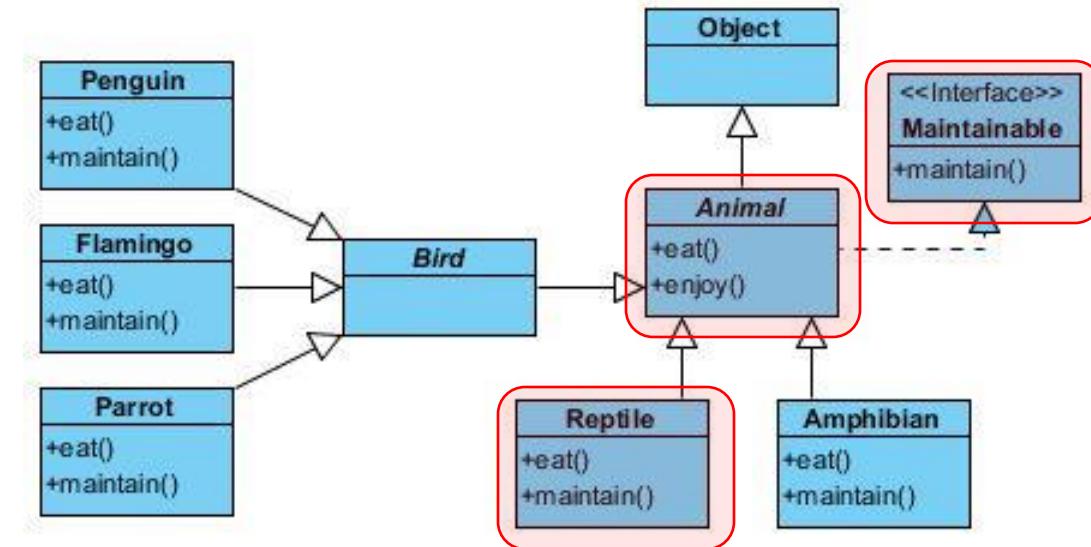
```
3  public class Reptile extends Animal {  
4      private int numTeeth;  
5  
6      public Reptile(String name, int numTeeth) {  
7          super(name);  
8          this.setNumTeeth(numTeeth);  
9      }  
10  
11     @Override  
12     public void eat() { System.out.println(this.getClass().getSimpleName()+" eats like a reptile."); }  
13  
14  
15     public int getNumTeeth() { return numTeeth; }  
16  
17  
18  
19  
20     public void setNumTeeth(int numTeeth) { this.numTeeth = numTeeth; }  
21  
22  
23  
24     @Override  
25     public void maintain() { System.out.println(this.getClass().getSimpleName()+" maintains life as reptile."); }  
26  
27  
28 }
```



Interfaces

</>

```
Run: Main ×  
C:\Program Files\Java\jdk-17\bin\java  
com.zooproject.Reptile@27973e9b  
Reptile eats like a reptile.  
Reptile enjoys life as animal.  
Reptile maintains life as reptile.  
  
com.zooproject.Reptile@506e1b77  
Reptile eats like a reptile.  
Reptile enjoys life as animal.  
Reptile maintains life as reptile.  
  
com.zooproject.Amphibian@4fca772d  
Amphibian eats like an amphibian.  
Amphibian enjoys life as amphibian.  
Amphibian maintains life as amphibian.  
  
Process finished with exit code 0
```



Lab Sheet has been released

questions?





Lecture 02B

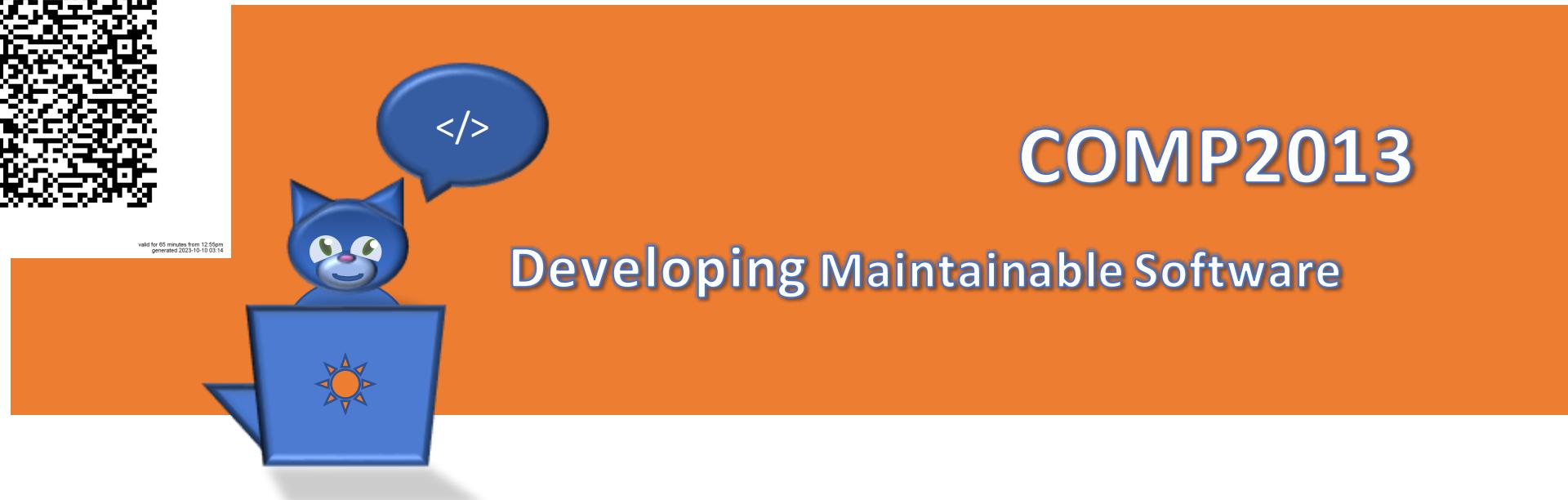
Horia A. Maior and Marjahan Begum

Lab Reflection + IntelliJ + Java Releases + jShell + Basic ZooApp Maintenance

COMP2013: Developing Maintainable Software
Week 3 – 1:00pm Thursday – 12 October 2023



valid for 65 minutes from 12:55pm
generated 2023-10-10 03:14



Lecture 02B

Lab Reflection + IntelliJ + Java Releases + jShell + Basic ZooApp Maintenance

Horia A. Maior and Marjahan Begum

Topics for this Week



- Lecture 02A:
 - OO and Java Refresher (2/2)
 - The missing bits from last lecture
 - Java Collections framework
 - Implementation of object oriented concepts in Java
- Lab 02:
 - Implementing the ZooApp
- Lecture 02B:
 - Lab reflection
 - IDEs + Java Releases
 - jShell
 - Maintaining the ZooApp (basic maintenance)

Software that we recommended you to install

</>

Software

- Full Installation Guidance (Windows) [[link](#)]
 - [Java 12+](#) (requires 64 bit system > use the virtual desktop if you don't have a 64 bit system)
 - [IntelliJ IDEA](#) (this is the IDE to be used for your coursework)
 - [Visual Paradigm](#) (also available as [online solution](#), but requires registration)
 - [JavaFX 12+](#) graphics library and [Gluon Scene Builder](#)
- It would be useful to use the latest versions, if you install the software on your private computers.



</>

valid for 65 minutes from 12:55pm
generated 2023-10-10 03:14

Labsheet 2



LAB 2: BUILDING THE ZOO EXAMPLE

Aims:

- Implement some of the object-oriented examples we saw in lecture 2A (to remind yourself, go back to lecture 2A slides)
- Gain more experience of object oriented programming
 - Remember that familiarity with this will not only help you write better code, but will aid understanding of existing projects

Coverage:

- In Part 1 (first half of this worksheet) we cover:
 - setters, getters, static variables, ArrayLists, and IDE shortcuts
- In Part 2 (second half of this worksheet) we cover:
 - Aggregation vs Composition
 - Writing the Employee classes for the Zoo example, including an abstract class and an interface

We release the labsheets on Mondays, for you to start working on this right away. This work goes beyond the lab session time, so we encourage you to finish this work, and use the lab time for support.

Questions channels on Teams. Tag one of our teaching support group in the questions channel. Ask generic questions on the peer support channel. We aim to respond as quickly as possible.

</>

intellij

Useful Material to Learn More about IntelliJ

</>

- Check out the "IntelliJ IDEA by JetBrains" channel for more
 - <https://www.youtube.com/channel/UC4ogdcPcIAOOMJktgBMhQnQ>
- IntelliJ IDEA Full Course 2020 (video 2h 35 min; skip first 23 min)
 - <https://www.youtube.com/watch?v=yefmcX57Eyg>
- TutorialsPoint: IntelliJ (text-based)
 - https://www.tutorialspoint.com/intellij_idea/index.htm

Code Completion

</>

- Code completion:
 - Start typing and use {TAB} key to use choice offered or {CTRL+.} to use the choice offered followed by a ":"
 - Type shortcuts and hit {TAB} key
 - For example: "sout" + {TAB} > "System.out.println();"
 - Hit {CTRL+J} to see all available shortcuts
 - Use clipboard viewer to find and reuse copied content, by pressing {CTRL+SHIFT+V}

Useful Plugins

</>

- Productivity Bundle Plugins: <https://plugins.jetbrains.com/bundles/4-productivity-bundle>
 - Key Promoter X: <https://plugins.jetbrains.com/plugin/9792-key-promoter-x>
 - IDE Feature Trainer: <https://plugins.jetbrains.com/plugin/8554-ide-features-trainer>
- Presentation Assistant (show keys pressed):
<https://plugins.jetbrains.com/plugin/7345-presentation-assistant> (requires MacOS Keymap plugin <https://plugins.jetbrains.com/plugin/13258-macos-keymap>)

what's new in Java (highlights)

What's new in Java (highlights)

</>

<https://www.codejava.net/java-se/java-se-versions-history>

- Java 9
 - Modules
 - jShell (REPL)
- Java 10
 - Local-Variable Type Inference (using var)
 - Enhancements for garbage collection and compilation
- Java 11
 - Removal of JavaFX (now an independent module)

What's new in Java (highlights)

</>

- Java 12
 - New garbage collector
 - New switch expression (preview)
- Java 13
 - Switch expressions (preview)
 - Text Blocks (preview)

```
1  private static String expressionWithArrow(int i) {  
2      return switch (i) {  
3          case 1, 2 -> "one or two";  
4          case 3 -> "three";  
5          default -> "smaller than one or more than three";  
6      };  
7  }
```

```
1 // Without Text Blocks  
2 String html = "<html>\n" +  
3         "  <body>\n" +  
4         "    <p>Hello, Escapes</p>\n" +  
5         "  </body>\n" +  
6         "</html>\n";  
7  
8 // With Text Blocks  
9 String html = """  
10        <html>  
11            <body>  
12                <p>Hello, Text Blocks</p>  
13            </body>  
14        </html>""";
```

What's new in Java (highlights)

</>

- Java 14
 - JDK Flight Recorder event streaming
 - Pattern matching
 - Switch expressions
- Java 15
 - Text blocks, hidden classes, a foreign-memory access API, the Z Garbage Collector
 - Previews of sealed classes, pattern matching, and records

What's new in Java (highlights)

</>

- Java 16
 - Concurrent thread-stack processing for garbage collection (JVM)
 - Support for C++ 14 language features (for the C++ JDK)
 - "Elastic Metaspace" capability to more quickly return unused class metadata memory (JVM)
 - Pattern matching for instanceof
 - Records (classes that act as transparent carriers of immutable data)
- Java 17
 - Standardises sealed classes and interfaces
 - Pattern matching for switch statements (Preview)
 - Deprecates the Security Manager and Applet APIs

What's new in Java (highlights)

</>

- Java 18
 - Simple Web Server
 - Code Snippets in Java API Documentation
 - Vector API (Third Incubator) and Foreign Function & Memory API (Second Incubator)
 - Pattern Matching for switch (Second Preview)
- Java 19
 - Structured Concurrency (Incubator)
 - Pattern Matching for switch (Third Preview)
 - Vector API (Fourth Incubator) and Foreign Function & Memory API (Preview)
 - Virtual Threads (Preview)
 - Record Patterns (Preview)

- Java 20
 - virtual threads,
 - a vector API proposal,
 - structured concurrency,
 - a foreign function and memory API,
 - record patterns,
 - pattern matching for switch statements and expressions

JDK 21

This release is the Reference Implementation of version 21 of the Java SE Platform, as specified by JSR 396 in the Java Community Process.

JDK 21 reached General Availability on 19 September 2023. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

The features and schedule of this release were proposed and tracked via the [JEP Process](#), as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Features

- 430: String Templates (Preview)
- 431: Sequenced Collections
- 439: Generational ZGC
- 440: Record Patterns
- 441: Pattern Matching for switch
- 442: Foreign Function & Memory API (Third Preview)
- 443: Unnamed Patterns and Variables (Preview)
- 444: Virtual Threads
- 445: Unnamed Classes and Instance Main Methods (Preview)
- 446: Scoped Values (Preview)
- 448: Vector API (Sixth Incubator)
- 449: Deprecate the Windows 32-bit x86 Port for Removal
- 451: Prepare to Disallow the Dynamic Loading of Agents
- 452: Key Encapsulation Mechanism API
- 453: Structured Concurrency (Preview)

JDK 21 will be a long-term support (LTS) release from most vendors. For a complete list of the JEPs integrated since the previous LTS release, JDK 17, please see [here](#).

What's new in Java (highlights)

</>

- For full details: Tech Geek Next
 - <https://www.techgeeknext.com/java/java11-features>
...
...
– <https://www.techgeeknext.com/java/java19-features>
...
...
– <https://developers.redhat.com/articles/2023/09/21/whats-new-developers-jdk-21>
- For full details: Wikipedia
 - https://en.wikipedia.org/wiki/Java_version_history

Adoption Poll

</>

- Which Version of Java Should You Use?

- It's important to note that even in 2022, many applications continue to run on Java 8 and Java 11. While Java 17 offers LTS, it's certainly not unusual if your application is using 11 or even 8.

<https://www.stackchief.com/blog/Which%20Version%20of%20Java%20Should%20You%20Use%3F>

Are you upgrading to Java 12? (Check all that apply)

I'm still using Java 8! (45%, 135 Votes)

I am waiting for the next long term release before I upgrade. (33%, 100 Votes)

Yes! I'm upgrading to Java 12. (14%, 42 Votes)

No, I am not upgrading to Java 12. (8%, 25 Votes)

Total Voters: 302

<https://jaxenter.com/java-12-adoption-poll-157164.html>

</>

jShell

Live Demo of jShell

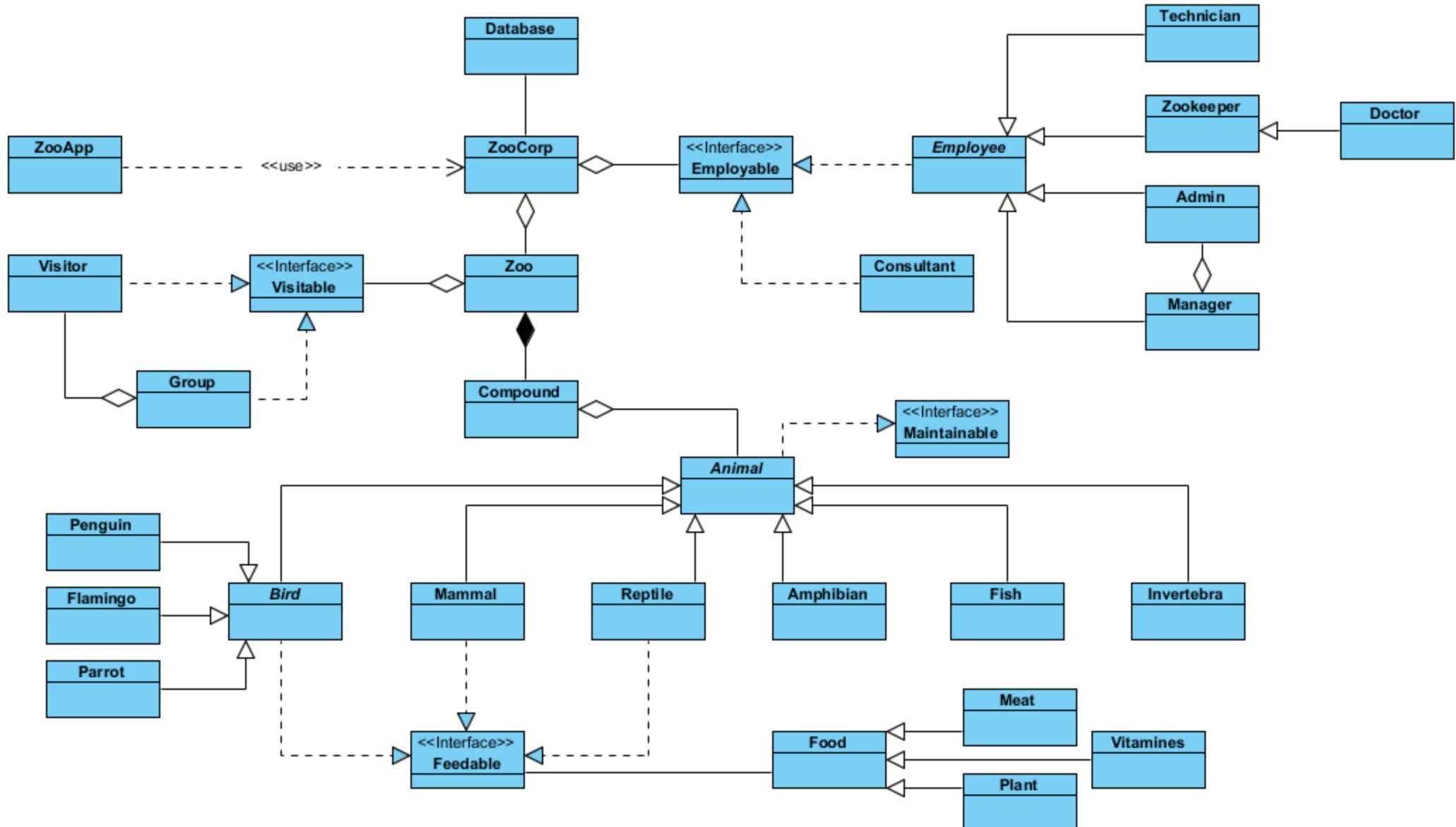
</>

```
Last login: Sun Oct  1 14:38:44 on ttys000
[psahm2@MACBOOK-DWHKGG46WW ~ % jShell
| Welcome to JShell -- Version 20.0.2
| For an introduction type: /help intro

jshell> 
```



basic ZooApp maintenance



Basic Maintenance – 2 min discussion in pairs

</>



- What would you propose how we should get started?



Refactoring code

</>

- Improved Code Quality
- Enhanced Readability
- Reduced Code Duplication
- Maintainability
- Scalability
- Performance Optimisation

- Goal
 - Improving current code without creating new functionality
 - Keep the code well organised and easy to maintain in the future
- Examples (see also Lecture 1A)
 - Adding unit tests
 - Renaming
 - Packaging
 - Avoiding code duplication
 - Comments and documentation
 - Moving code to where it belongs
 - Using correct access modifiers (encapsulation)

- ZooApp Development
 - Make ArrayLists private
 - Make all methods/classes used by the constructor "final"
 - Remove obsolete "this" keywords
 - Consider what the interface to the outside world should look like
 - Remove redundant setters and getters
 - Check the module-info.java file

Basic Maintenance

- Help from IntelliJ

Code	Refactor	Build	Run	Tools	VCS	Window	Help
Override Methods...							Ctrl+O
Implement Methods...							Ctrl+I
Delegate Methods...							
Generate...							Alt+Insert
Code Completion							>
Inspect Code...							
Code Cleanup...							
Analyze Code							>
Analyze Stack Trace or Thread Dump...							
Insert Live Template...							Ctrl+J
Save as Live Template...							
Surround With...							Ctrl+Alt+T
Unwrap/Remove...							Ctrl+Shift+Delete
Folding							>
Comment with Line Comment							Ctrl+/*
Comment with Block Comment							Ctrl+Shift+/*
Reformat Code							Ctrl+Alt+L
Reformat File...							Ctrl+Alt+Shift+L
Auto-Indent Lines							Ctrl+Alt+I
Optimize Imports							Ctrl+Alt+O
Rearrange Code							
Move Statement Down							Ctrl+Shift+Down
Move Statement Up							Ctrl+Shift+Up
Move Element Left							Ctrl+Alt+Shift+Left
Move Element Right							Ctrl+Alt+Shift+Right
Move Line Down							Alt+Shift+Down
Move Line Up							Alt+Shift+Up
Update Copyright...							
Generate module-info Descriptors							
Convert Java File to Kotlin File							Ctrl+Alt+Shift+K

</>

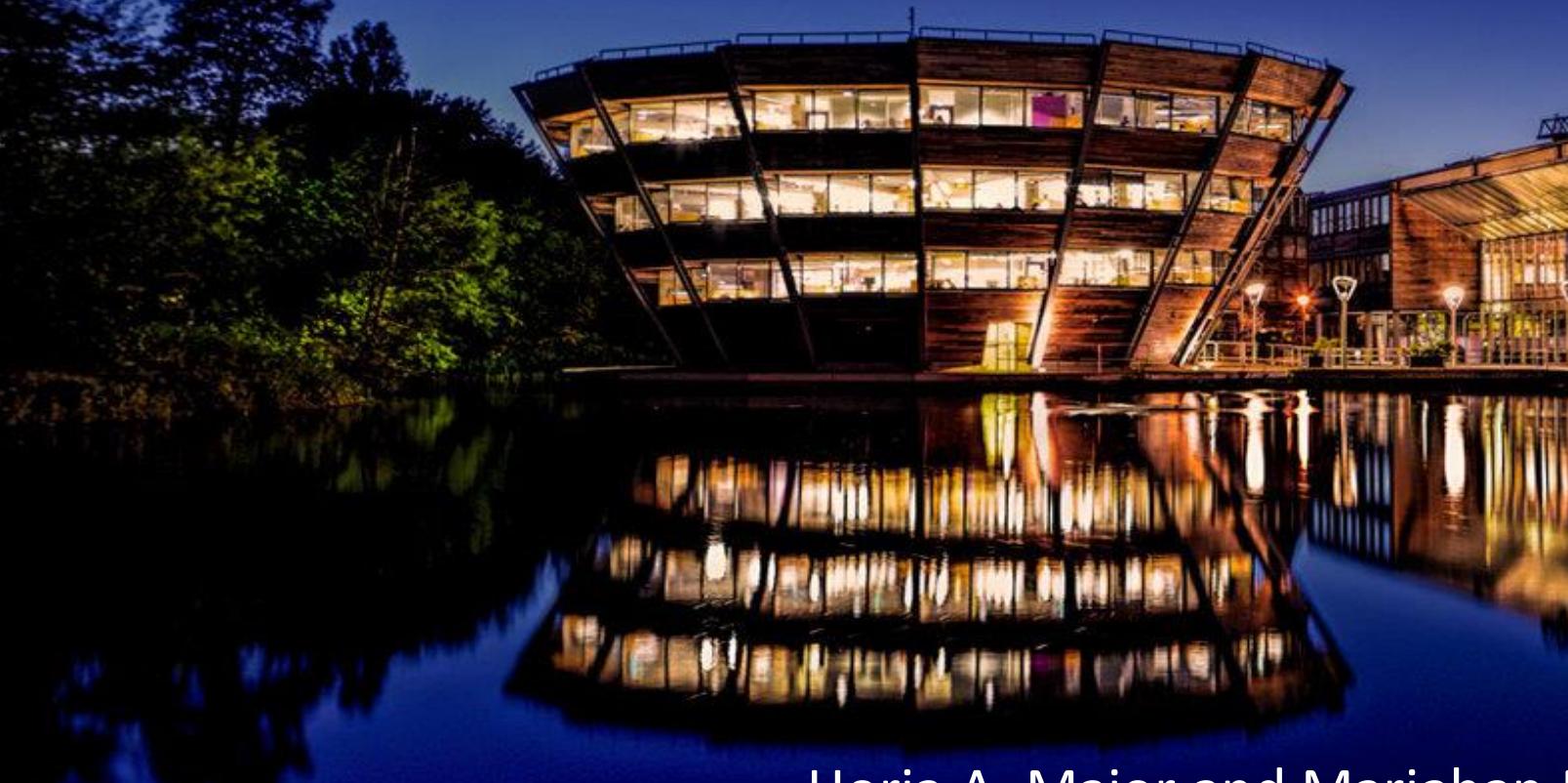
Some final remarks ...



Lecture 03A

Maintainable GUI Development (1/2)

Introduction to GUI Programming in Java



COMP2013: Developing Maintainable Software
Week 4 – 4:00pm Monday – 16 October 2023



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

Horia A. Maior and Marjahan Begum



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



Lecture 03A

Maintainable GUI Development (1/2)

Introduction to GUI Programming in Java

Horia A. Maior and Marjahan Begum

What's coming up ...



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

Teaching Week	Topic
1	Introduction to DMS
2	More Advanced Java Topic
3	Maintainable GUI Development (1/2)
4	Design Principles and Patterns
5	Maintainable GUI Development (2/2)
6	Coding and Repository Tools for DMS
7	UML for the Maintainer
8	Refactoring Skills.
9	Open Source
10	Guest Lecture
11	Revision and Exam Prep

Coursework 1 released

Coursework 1 Part 1 Deadline

Coursework 1 Part 2 Deadline

Topics for this Week



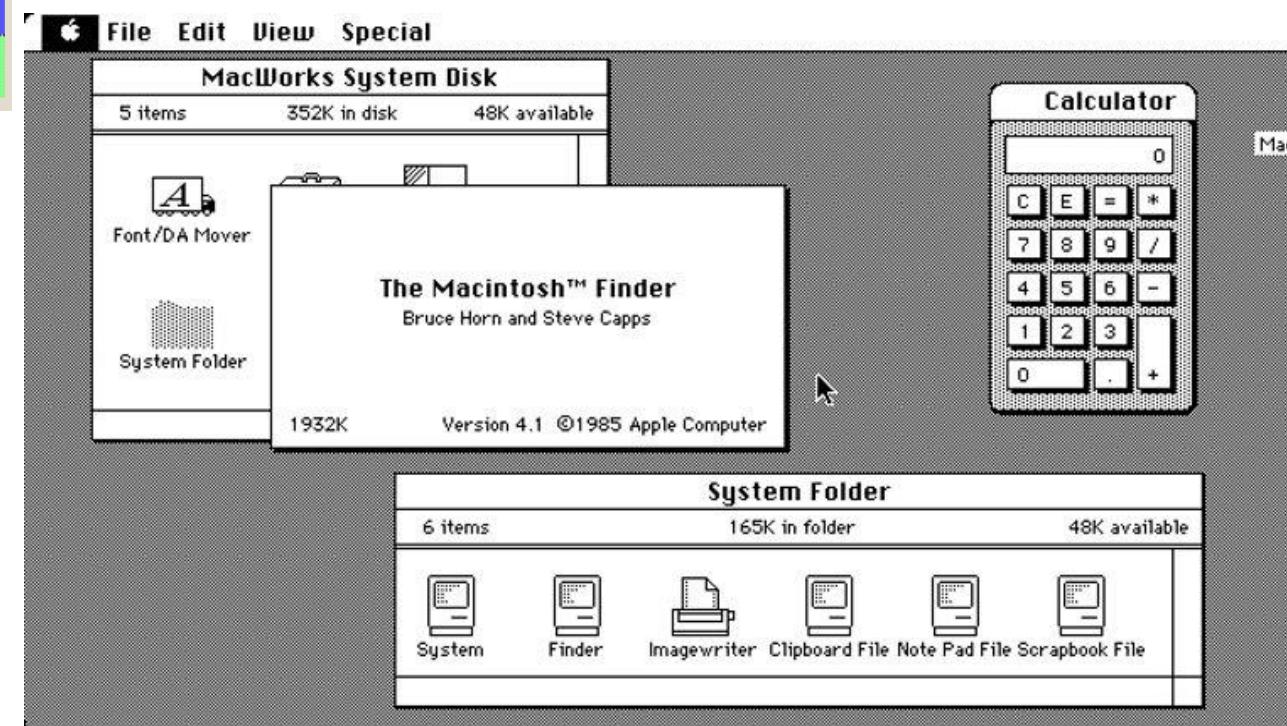
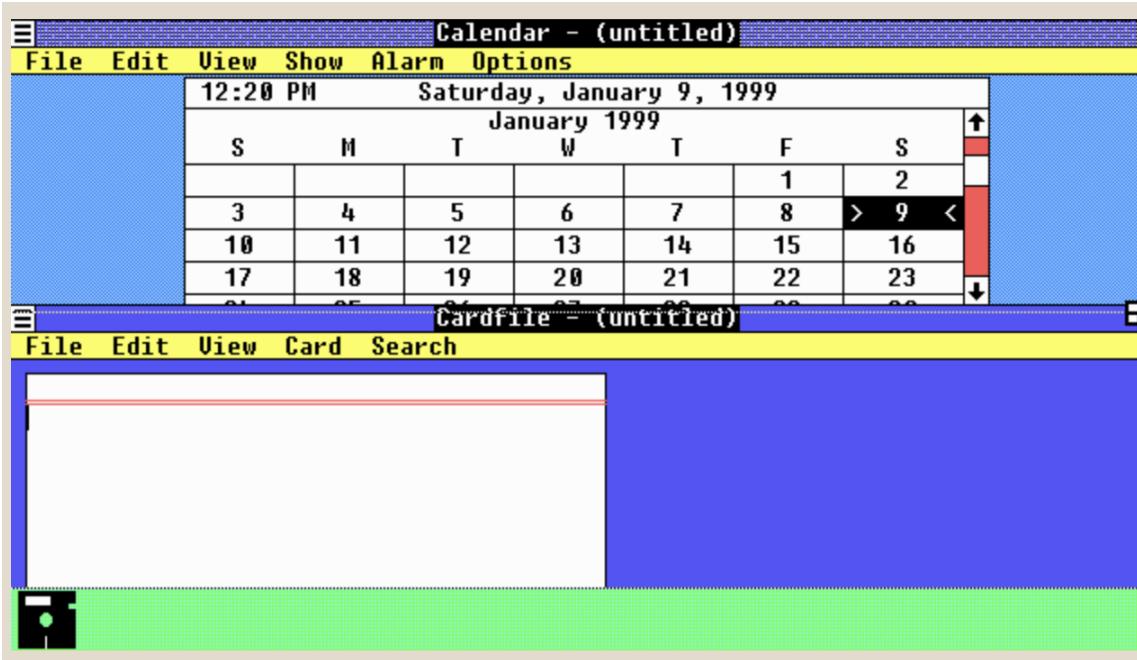
valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14

- Lecture 03A
 - Introduction to GUI programming in Java
 - GUIs; Swing vs JavaFX; JavaFX foundations
- Lab 03
 - Practice JavaFX GUI development (phone app)
- Lecture 03B
 - GUI development in practice
 - Lab reflection; MVC pattern revisited; JavaFX and FXML



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

Graphical User Interfaces (GUIs)



<http://www.catb.org/~esr/writings/taouu/html/ch02s05.html>

What are GUIs?

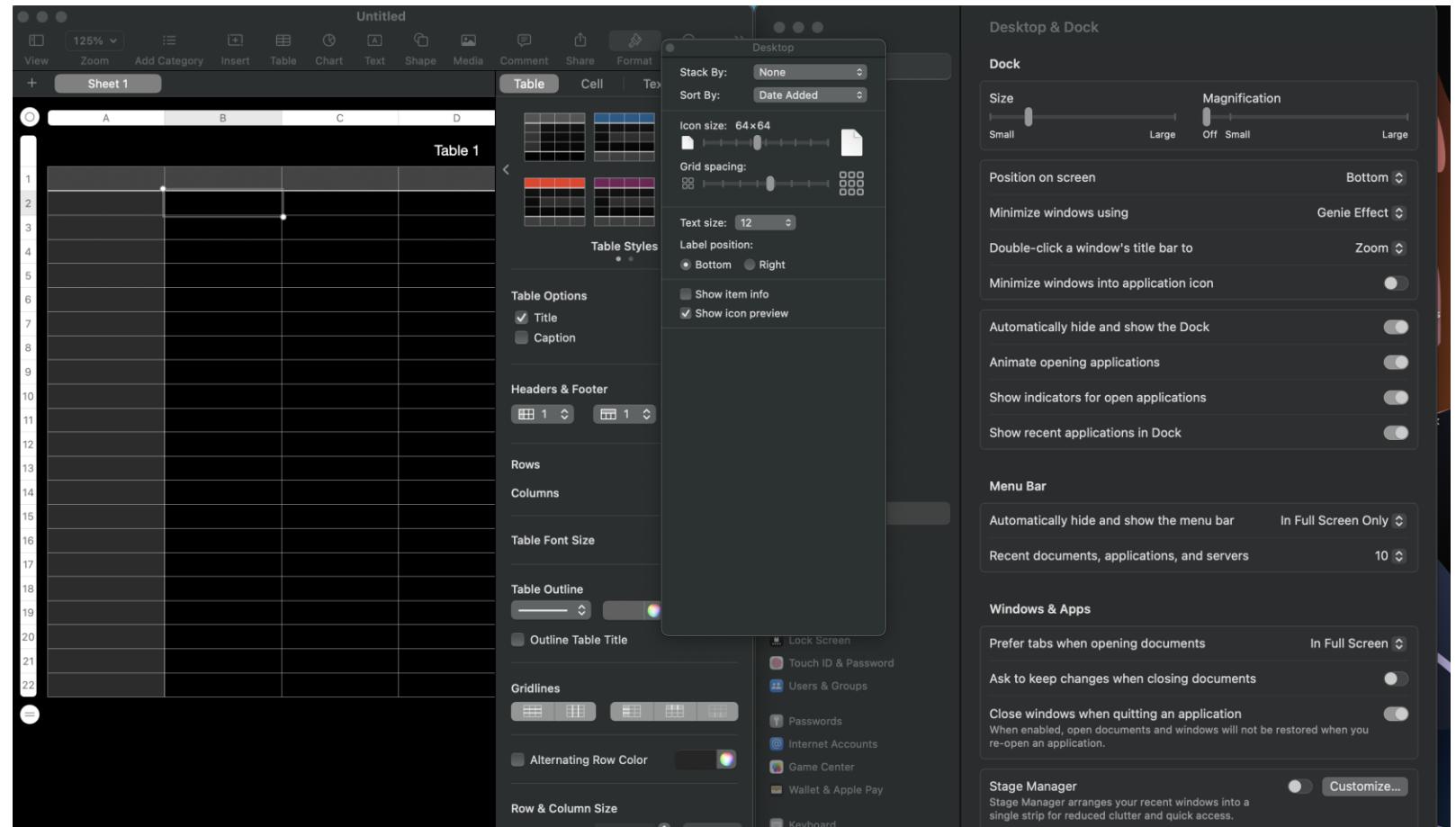


- **What are Graphical User Interfaces (GUIs)?**
 - GUIs are a form of user interface that allow users to interact with electronic devices through graphical icons [...] as primary notation, instead of text-based user interfaces, typed command labels or text navigation. [\[Wikipedia\]](#)
 - GUIs were introduced in reaction to the perceived steep learning curve of Command Line Interfaces (CLIs) which require commands to be typed on a computer keyboard. [\[Wikipedia\]](#)
- Why do I need to learn about them in this module?
 - Need to understand the link between a GUI and the code behind it in order to maintain it
 - Want to write new GUIs from scratch that are easy to maintain

What are GUIs

</>

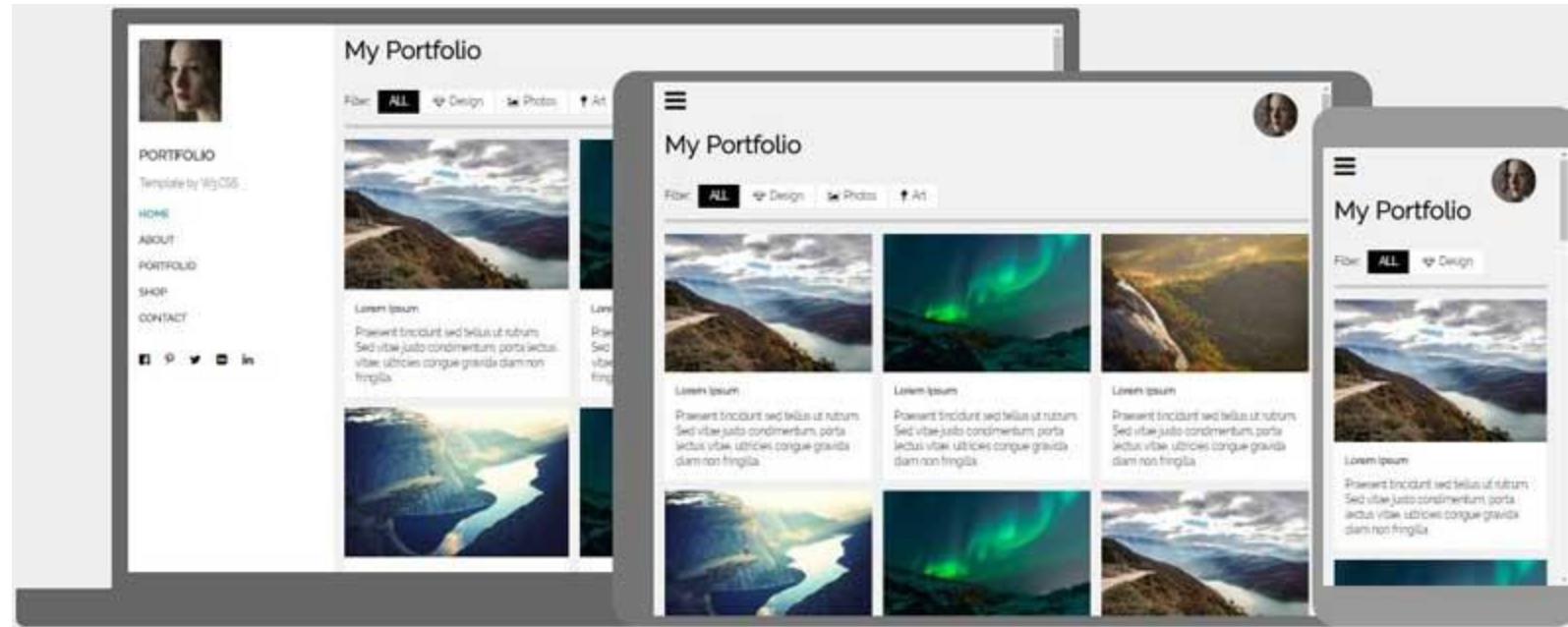
- GUIs can often be set up in different ways depending on personal needs and likes



What are GUIs?

</>

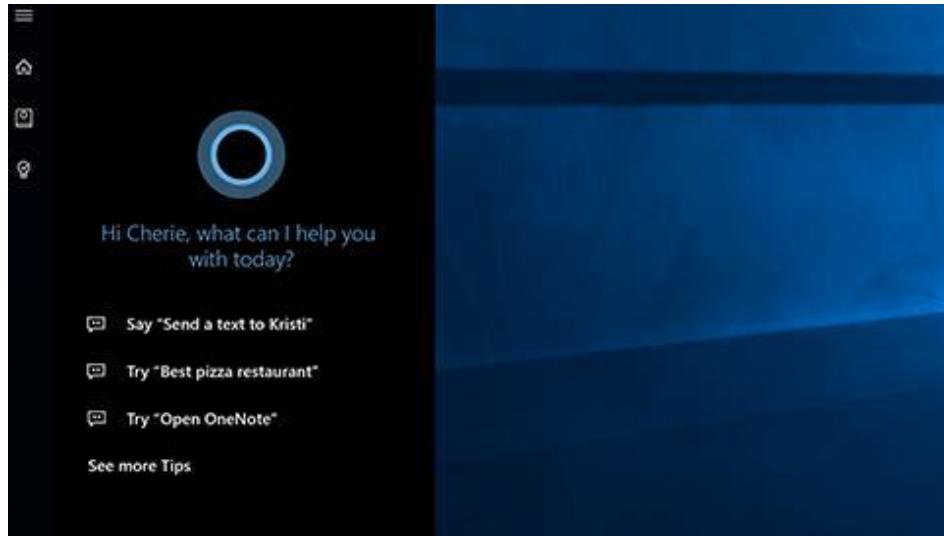
- Different devices require different GUI design styles: We need responsive GUIs



What are GUIs?

</>

- Does the GUI have a future?



Standards and Guidance

</>

ISO 9241 defines usability as effectiveness, efficiency, and satisfaction with which users accomplish tasks

<https://www.iso.org/obp/ui/#iso:std:iso:9241:-161:en>

Online Browsing Platform (OBP)

Search ISO 9241-161:2016(en) ×

ISO 9241-161:2016(en) Ergonomics of human-system interaction — Part 161: Guidance on visual user-interface elements

Table of contents

Foreword

Introduction

1 Scope

2 Normative references

3 Terms and definitions

4 Accessibility

5 Relationship of input methods and visual user-interface elements

6 States of visual user-interface elements

7 Describing visual user-interface elements

8 Visual user-interface elements

- 8.1 Accordion
- 8.2 Analogue form element/slider
- 8.3 Carousel/Carousel
- 8.4 Check box/check button
- 8.5 Collapsible container
- 8.6 Colour picker
- 8.7 Combination box/combo box
- 8.8 Cursor
- 8.9 Date picker
- 8.10 Dialogue box
- 8.11 Dropdown list box
- 8.12 Entry field/input field
- 8.13 Entry field with dialogue button
- 8.14 Geographical map
- 8.15 Group/group box
- 8.16 Handle
- 8.17 Hierarchical list/tree view/tree list
- 8.18 Implicit designator
- 8.19 Instructive information
- 8.20 Input tokenizer
- 8.21 Label
- 8.22 Legend/chart key

Available in: EN FR

Introduction

In different communities in the interactive system development ecosystem, the use, the names and the understanding of user-interface elements differs significantly. One of the results is that users have to cope with elements which differ in terms of keyboard entry and control, mouse behaviour, visual presentation of functionality and different options to control elements. **Consistent element behaviour, functionality and rendering is crucial for the usability of user interfaces.** This causes added efforts in all stakeholders in human-centred design activities, since this multitude needs to be managed in order to ensure high-quality collaboration of various specialists. Especially in the light of new emerging user-interface concepts and designs, a common definition of visual user-interface elements and the rationale for their selection, as well as their use can be regarded as an effort to sustain cooperation and ensure a sound basis for professional conversation. It is also of importance to state that this part of ISO 9241 of visual user-interface elements in no ways predetermines a visual style of the elements themselves, thus avoiding to impress determinants in creation, brand usage and style development. In addition, this part of ISO 9241 is laid out in an independent of platform specifics, so that no specific industrial user-interface styleguide, implementation technology or development process needs to be observed in order to be compliant with this part of ISO 9241.

This part of ISO 9241 aims to provide information on visual user-interface elements to help those responsible for managing software design and re-design processes, create user interface specifications, styleguides and visual concepts to identify, plan and design effective, efficient and satisfactory interactive systems.

Visual user-interface elements described in this part of ISO 9241 complements existing systems design approaches, methods or processes. They can be referenced in any kind of user interface strategy, regardless of the technology used for the user interface.

Table 1 — Overview of different visual user interface properties that are used to build a user interface design

User Interface Design				
Interactive Properties		Informative Properties		Decorative Properties



- Guiding principles for designing high quality GUIs?
 1. Keep it simple
 2. Create consistency and use common elements
 3. Be purposeful in page layout
 4. Strategically use colour and texture
 5. Use typography to create hierarchy and clarity
 6. Make sure that the system communicates what's happening
 7. Think about the defaults

<https://www.usability.gov/what-and-why/user-interface-design.html>

Heuristics and Guidelines

</>

- <https://www.nngroup.com/articles/ten-usability-heuristics/>

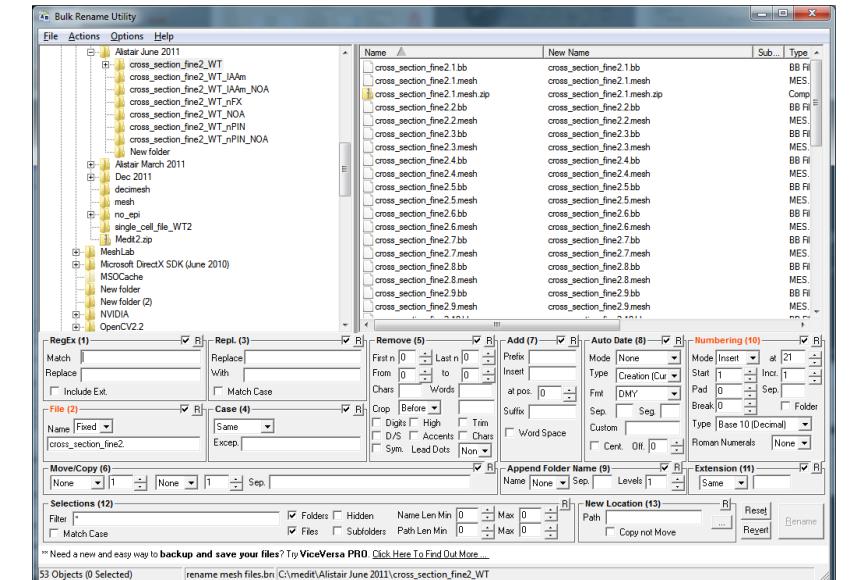
Nielsens (1993), 10 usability heuristics

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help recognize, diagnose, and recover from errors
10. Help and documentation

</>

Examples of Bad UI Design

- Appearance over functionality
- What does the user need to do?
- Accessibility of controls
- Form field overload



COMP2013

basics of GUI programming

Swing and JavaFX

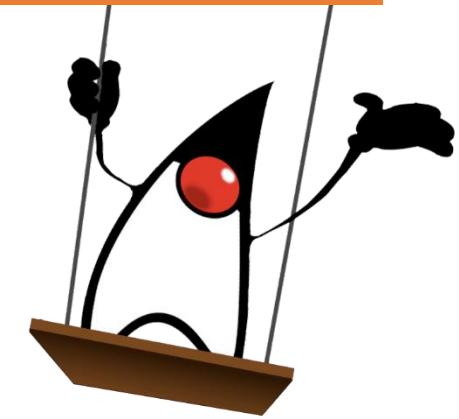
What do I need to know before getting started?

</>

- Under the hood GUIs are event-driven
 - The user decides the order of execution depending whether they click a button, or select a drop-down, or choose a menu item etc.
 - The system is waiting for something to happen
- GUIs are implemented by using frameworks or libraries
 - Java examples: Swing + JavaFX
- GUIs can be programmatically designed or drawn in a graphical editor

Swing

</>



- Java's first attempt at a 'modern' GUI approach
 - Came about in the late 1990's...
 - Superseded/built on the earlier AWT (Abstract Window Toolkit)
 - Swing still relies on AWT for some circumstances
 - You can spot a Swing component as it starts with a J, e.g. JFrame
 - Many existing Java GUIs use Swing, and it still has a strong following
 - But Swing is being slowly retired in favour of JavaFX

Java Swing Example

</>

- Hello World GUI in Java and Swing
- Video Recording Demo Available:
 - [Demo Hello World Java Swing - HD 1080p.mov](#)

Module java.desktop

Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.

Module Graph:



DEMO

<https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/module-summary.html>

```
module HelloWorldSwingGUI {  
    opens com.COMP2013;  
    requires java.desktop;  
}
```

</>

DEMO

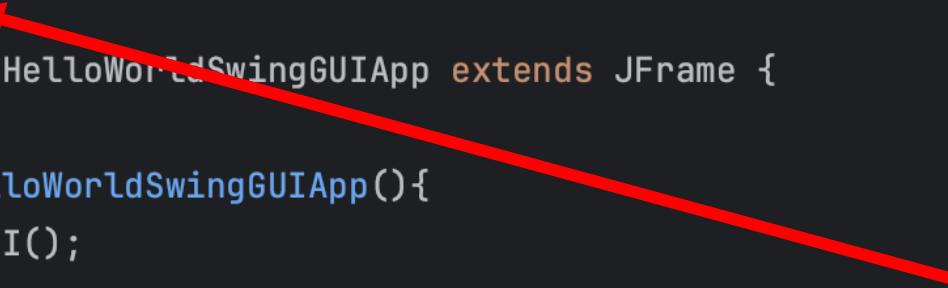
```
1 package com.COMP2013;
2 > import ...
6 ▷ public class HelloWorld Swing GUI App extends JFrame {
7     1 usage
8     public HelloWorld Swing GUI App(){
9         initUI();
10    }
11    1 usage
12    private void initUI(){
13        setSize( width: 600, height: 600);
14        setTitle("My first GUI in Java Swing. Hello world!");
15        setLocationRelativeTo(null);
16        setDefaultCloseOperation(EXIT_ON_CLOSE);
17        JPanel pannel = new JPanel(new GridBagLayout());
18        JButton button1 = new JButton( text: "Exit program");
19        button1.addActionListener(new ActionListener() {
20            @Override
21            public void actionPerformed(ActionEvent e) {
22                System.out.println("Now we will exit the program");
23                System.exit( status: 0);
24            }
25        });
26        pannel.add(button1);
27        this.add(pannel);
28    }
```

module-info.java (HelloWorld Swing GUI) ×

```
1 module HelloWorld Swing GUI {
2     opens com.COMP2013;
3     requires java.desktop;
4 }
```

</>

DEMO

```
1 package com.COMP2013;
2 > import ... 
3
4 D public class HelloWorld Swing GUI App extends JFrame {
5
6     1 usage
7     public HelloWorld Swing GUI App(){
8         initUI();
9     }
10    1 usage
11    private void initUI(){
12        setSize( width: 600 , height: 600 );
13        setTitle("My first GUI in Java Swing. Hello world!");
14        setLocationRelativeTo(null);
15        setDefaultCloseOperation(EXIT_ON_CLOSE);
16        JPanel pannel = new JPanel(new GridBagLayout());
17        JButton button1 = new JButton( text: "Exit program");
18        button1.addActionListener(new ActionListener() {
19            @Override
20            public void actionPerformed(ActionEvent e) {
21                System.out.println("Now we will exit the program");
22                System.exit( status: 0 );
23            }
24        });
25        pannel.add(button1);
26        this.add(pannel);
27    }
28 }
```

```
3     import javax.swing.*;
4     import java.awt.*;
5     import java.awt.event.ActionEvent;
6     import java.awt.event.ActionListener;
```

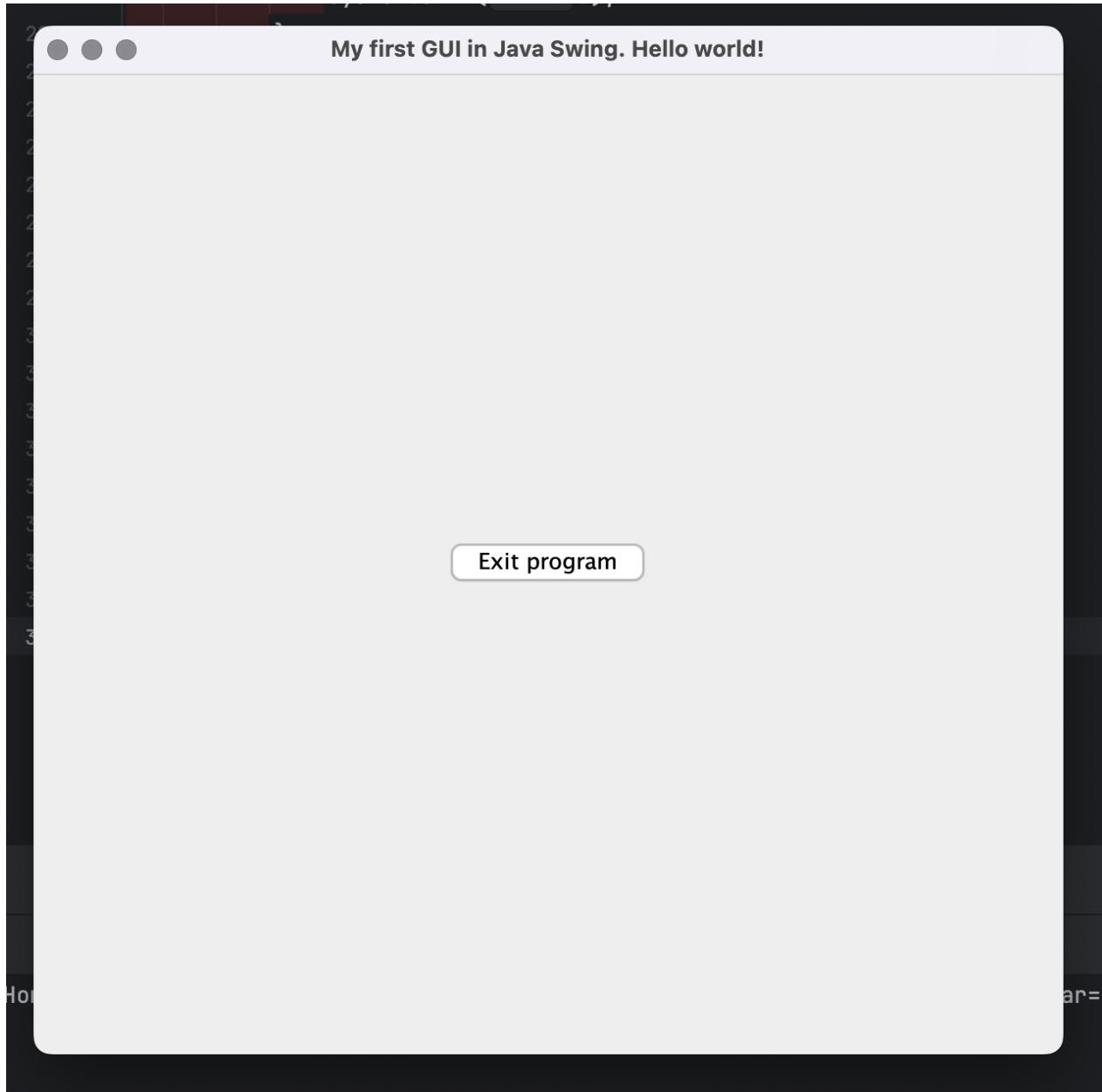
```
1 package com.COMP2013;
2 > import ...
3
4 D public class HelloWorldSwingGUIApp extends JFrame {
5     1 usage
6     public HelloWorldSwingGUIApp(){
7         initUI();
8     }
9     1 usage
10    private void initUI(){
11        setSize( width: 600, height: 600);
12        setTitle("My first GUI in Java Swing. Hello world!");
13        setLocationRelativeTo(null);
14        setDefaultCloseOperation(EXIT_ON_CLOSE);
15        JPanel pannel = new JPanel(new GridBagLayout());
16        JButton button1 = new JButton( text: "Exit");
17        button1.addActionListener(new ActionListener{
18            @Override
19            public void actionPerformed(ActionEvent e) {
20                System.out.println("Now we will exit");
21                System.exit( status: 0);
22            }
23        });
24        pannel.add(button1);
25        this.add(pannel);
26    }
```

```
module-info.java (HelloWorldSwingGUI) ×
1 module HelloWorldSwingGUI {
2     opens com.COMP2013;
3     requires java.desktop;
```

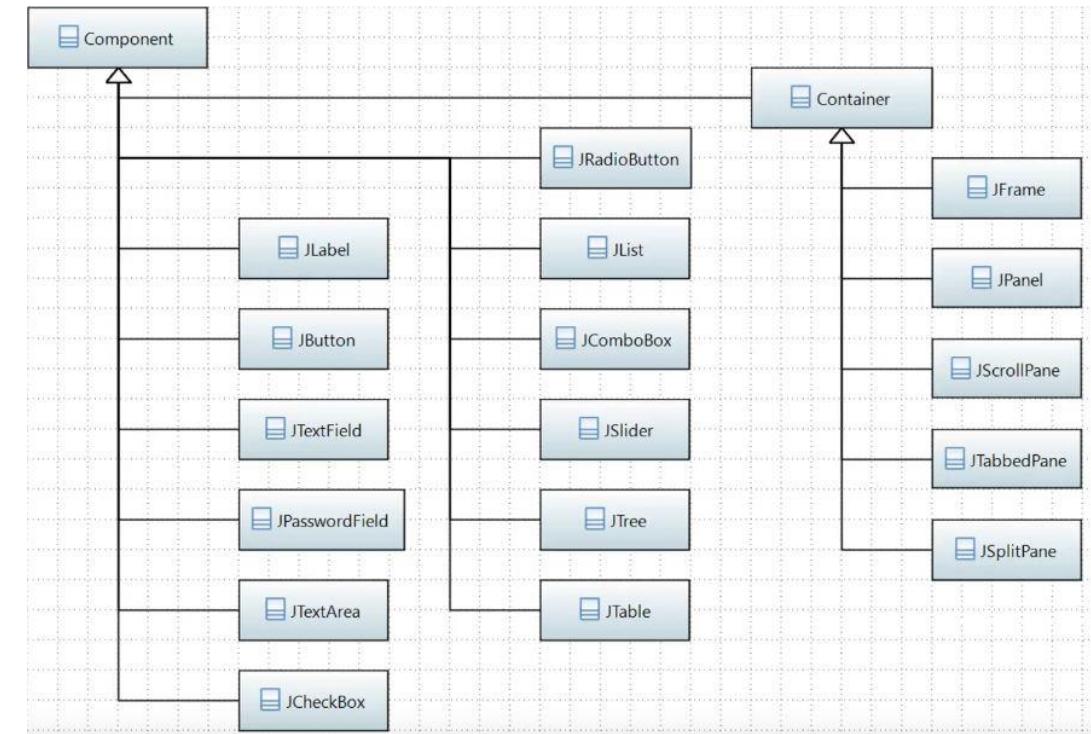
```
27
28
29 D public static void main(String[] args) {
30     //main function
31     EventQueue.invokeLater(new Runnable() {
32
33     @Override
34     public void run() {
35         new HelloWorldSwingGUIApp().setVisible(true);
36     }
37 }
38 }
```

</>

DEMO



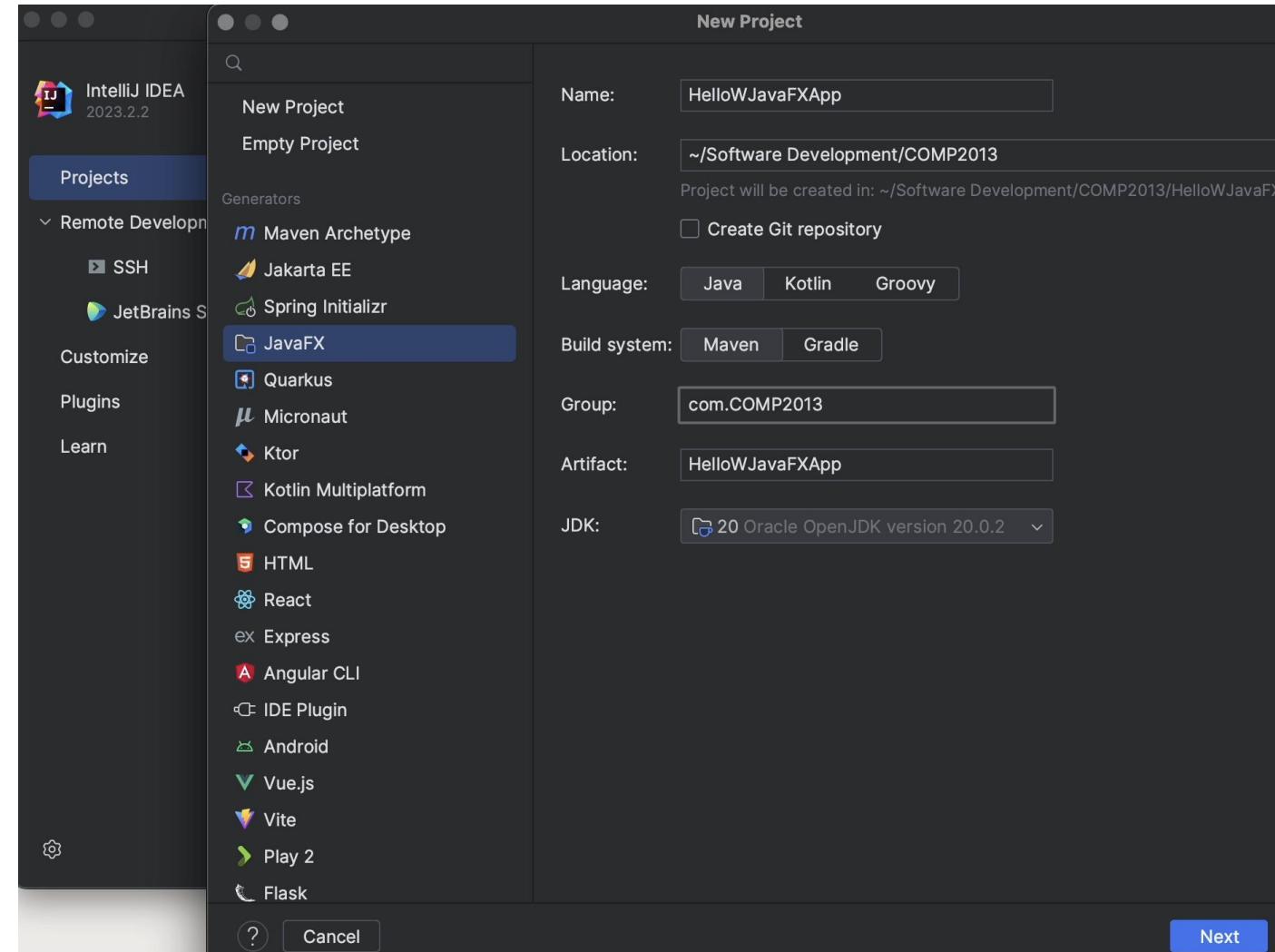
- For those of you not (so) familiar with Swing, here are some excellent tutorials:
 - <https://www.youtube.com/playlist?list=PLTMybUaeagJagT2qoftaf5CkCvgc3pBTn>



- More recently introduced as part of JDK/JRE (until Java 10) but then removed from the JDK/JRE (from version 11 onwards) and available as external library
 - Can deploy GUIs to tablet, phone, desktop etc.
 - Able to separate GUI code from program code using an XML description file (FXML)
 - Handling graphics by using a hardware-accelerated graphics pipeline to do the rendering job
 - Supports idea of properties (variables that represent the state of an instantiated object)
 - Formatting uses Cascading Style Sheets (CSS)
 - Special effects/animations supported
 - Comes with media player



</>



</>

DEMO

```
1 module com.comp2013 {  
2     opens com.comp2013;  
3     requires javafx.controls;  
4     requires javafx.graphics;  
5 }
```

</>

DEMO

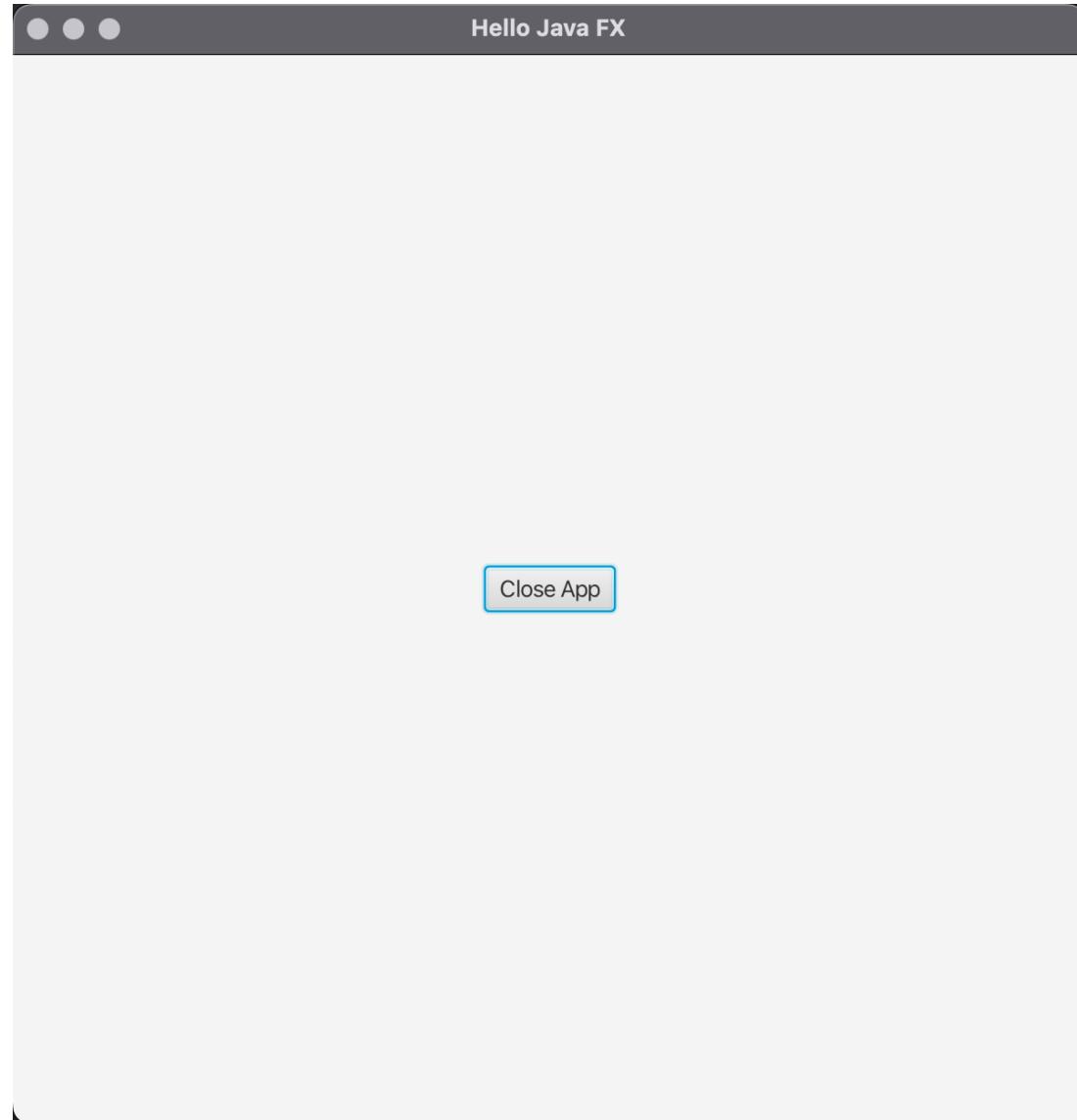
```
1 package com.comp2013;
2
3 > import ...
4
5
6
7
8
9
10
11
12 ▶ public class HelloApplication extends Application {
13
14     @Override
15     @
16         public void start(Stage stage) throws IOException {
17             stage.setTitle("Hello Java FX");
18             Button button1 = new Button(s: "Close App");
19             button1.setOnAction(new EventHandler<ActionEvent>() {
20                 @Override
21                     public void handle(ActionEvent actionEvent) {
22                         System.out.println("Bye Bye!");
23                         System.exit(status: 0);
24                     }
25             });
26             StackPane root = new StackPane();
27             root.getChildren().add(button1);
28             stage.setScene(new Scene(root, v: 600, v1: 600));
29             stage.show();
30         }
31         public static void main(String[] args) {
32             launch();
33         }
34 }
```

```
1 package com.comp2013;
2
3 > import ...
12
13 D public class HelloApplication extends Application {
14     @Override
15     public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button("Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, 600, 600));
28         stage.show();
29     }
30     public static void main(String[] args) {
31         launch();
32     }
33 }
```

3 import javafx.application.Application;
4 import javafx.event.ActionEvent;
5 import javafx.event.EventHandler;
6 import javafx.scene.Scene;
7 import javafx.scene.control.Button;
8 import javafx.scene.layout.StackPane;
9 import javafx.stage.Stage;

</>

DEMO



Adding Framework Support ...

</>

DEMO

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "HelloJavaFXWorld". It includes a src folder containing main and test packages, each with Java files (HelloJavaFXWorldApp.java and HelloJavaFXWorldAppTest.java) and module-info.java. There are also .iml and pom.xml files.
- Maven Tool Window:** Located on the right, it displays the Maven project tree for "hellojavafxworld". It shows the following structure:
 - Lifecycle:** clean, validate, compile, test, package, verify, install, site, deploy
 - Plugins:** clean (org.apache.maven.plugins:maven-clean-plugin:2.5), compiler (org.apache.maven.plugins:maven-compiler-plugin), deploy (org.apache.maven.plugins:maven-deploy-plugin:2), install (org.apache.maven.plugins:maven-install-plugin:2.4), jar (org.apache.maven.plugins:maven-jar-plugin:2.4), javafx (org.openjfx:javafx-maven-plugin:0.8). Sub-items under javafx include javafx:link and javafx:run.
 - Dependencies:** resources (org.apache.maven.plugins:maven-resources-plugin:3.0.0), site (org.apache.maven.plugins:maven-site-plugin:3.3), surefire (org.apache.maven.plugins:maven-surefire-plugin:2.18.1).
- Code Editor:** The central editor shows the content of the pom.xml file. The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.siebers</groupId>
    <artifactId>hellojavafxworld</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <junit.version>5.8.2</junit.version>
        <javafx.version>17.0.2</javafx.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M6</version>
            </plugin>
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.8</version>
                <executions>
                    <execution>
                        <!-- Default configuration for running with: mvn clean javafx:run -->
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

```

The bottom status bar shows the time as 32:34, encoding as LF, and file as UTF-8 with 4 spaces.

getting started with JavaFX



```
1 package com.comp2013;
2
3 > import ...
12
13 > public class HelloApplication extends Application {
14
15     @Override
16     public void start(Stage stage) throws IOException {
17         stage.setTitle("Hello Java FX");
18         Button button1 = new Button(s: "Close App");
19         button1.setOnAction(new EventHandler<ActionEvent>() {
20             @Override
21             public void handle(ActionEvent actionEvent) {
22                 System.out.println("Bye Bye!");
23                 System.exit(status: 0);
24             }
25         });
26         StackPane root = new StackPane();
27         root.getChildren().add(button1);
28         stage.setScene(new Scene(root, v: 600, v1: 600));
29         stage.show();
30     }
31
32     public static void main(String[] args) {
33         launch();
34     }
35 }
```

launch readies the application and then invokes **start**

Execution moves then to the JavaFX thread

JavaFX Theatre analogy

</>

The whole play
contains several *Scenes*



All the action *Scenes*
takes place on a *Stage*

JavaFX's Stage and Scene

</>

- Stage:
 - Think of it as an application window
 - Depending on OS, there may be only one
 - Equivalent to Swing's JFrame (or JDialog)
- Scene:
 - Equivalent to a content pane
 - Holds other objects (JavaFX Node objects)

```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     public void start(Stage stage) throws IOException {
16         stage.setTitle("Hello Java FX");
17         Button button1 = new Button("Close App");
18         button1.setOnAction(new EventHandler<ActionEvent>() {
19             @Override
20             public void handle(ActionEvent actionEvent) {
21                 System.out.println("Bye Bye!");
22                 System.exit(0);
23             }
24         });
25         StackPane root = new StackPane();
26         root.getChildren().add(button1);
27         stage.setScene(new Scene(root, 600, 600));
28         stage.show();
29     }
30     ▶ public static void main(String[] args) {
31         launch();
32     }
33 }
```

So this is our window,
and we can set its text
in the title bar etc.

```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @
16         public void start(Stage stage) throws IOException {
17             stage.setTitle("Hello Java FX");
18             Button button1 = new Button("Close App");
19             button1.setOnAction(new EventHandler<ActionEvent>() {
20                 @Override
21                     public void handle(ActionEvent actionEvent) {
22                         System.out.println("Bye Bye!");
23                         System.exit(0);
24                     }
25             StackPane root = new StackPane();
26             root.getChildren().add(button1);
27             stage.setScene(new Scene(root, 600, 600));
28             stage.show();
29         }
30     ▶ public static void main(String[] args) {
31         launch();
32     }
33 }
```

These are examples of
Node objects

```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @
16         public void start(Stage stage) throws IOException {
17             stage.setTitle("Hello Java FX");
18             Button button1 = new Button(s: "Close App");
19             button1.setOnAction(new EventHandler<ActionEvent>() {
20                 @Override
21                     public void handle(ActionEvent actionEvent) {
22                         System.out.println("Bye Bye!");
23                         System.exit(status: 0);
24                     }
25             });
26             StackPane root = new StackPane();
27             root.getChildren().add(button1);
28             stage.setScene(new Scene(root, v: 600, v1: 600));
29             stage.show();
30     }
31     public static void main(String[] args) {
32         launch();
33     }
```

Node objects can contain other **Node** objects

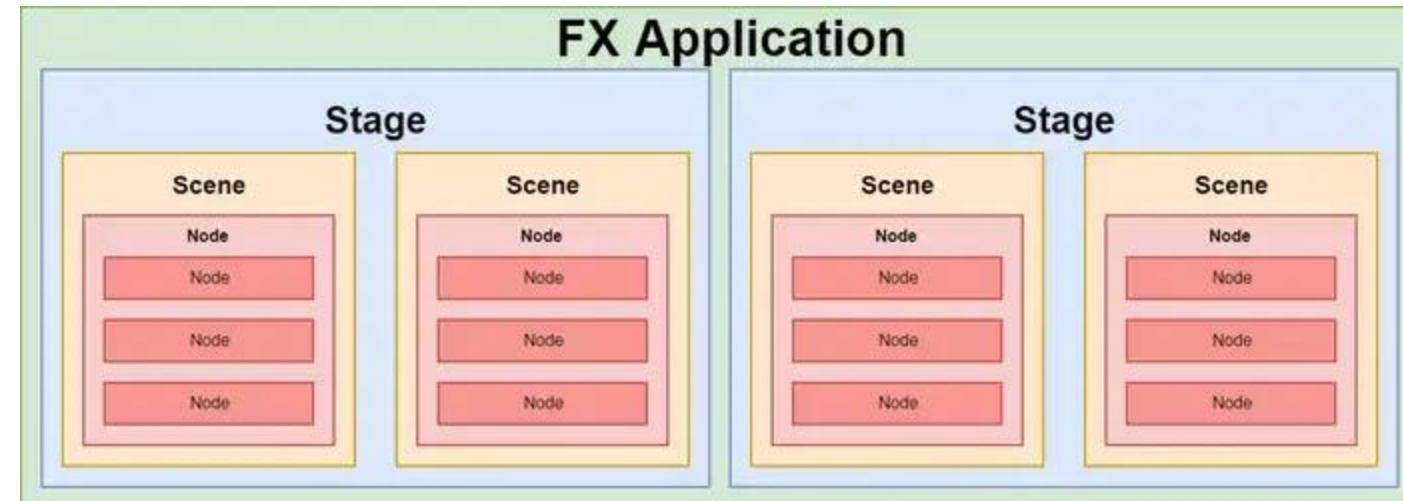
```
1 package com.comp2013;
2
3 > import ...
12
13 ▶ public class HelloApplication extends Application {
14     @Override
15     @
16         public void start(Stage stage) throws IOException {
17             stage.setTitle("Hello Java FX");
18             Button button1 = new Button(s: "Close App");
19             button1.setOnAction(new EventHandler<ActionEvent>() {
20                 @Override
21                     public void handle(ActionEvent actionEvent) {
22                         System.out.println("Bye Bye!");
23                         System.exit(status: 0);
24                     }
25             });
26             StackPane root = new StackPane();
27             root.getChildren().add(button1);
28             stage.setScene(new Scene(root, v: 600, v1: 600));
29             stage.show();
30     }
31     public static void main(String[] args) {
32         launch();
33     }
```

Finally, we put it all together,
and make it visible

Summary: Stage / Scene / Node

</>

- Summary
 - If the OS allows each application can have multiple stages - windows. Each stage can switch between multiple scenes. Scenes contain nodes - layout and regular components.



From: <https://www.vojtechruzicka.com/javafx-hello-world/>

Properties & Binding

</>

- JavaFX properties (observable containers) are often used in conjunction with binding, a powerful mechanism for expressing direct relationships between variables
- When objects participate in bindings, changes made to one object will automatically be reflected in another object
- You can also add a change listener to be notified when the property's value has changed

```
1 package com.COMP2013;
2 > import ...
3
4 D public class HelloApplication extends Application {
5
6     @Override
7     public void start(Stage stage) throws IOException {
8         stage.setTitle("Welcome to the PropertyBinding Demo!");
9         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
10        Text text = new Text(String.valueOf((int)slider.getValue()));
11        text.setFont(Font.font(s: "Verdana", v: 50));
12        StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
13        slider.valueProperty().addListener(new ChangeListener<Number>() {
14            @Override
15            public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
16                text.setText(String.valueOf((int)slider.getValue()));
17            }
18        });
19        StackPane root = new StackPane();
20        root.getChildren().addAll(slider, text);
21        stage.setScene(new Scene(root, v: 500, v1: 500));
22        stage.show();
23    }
24
25    public static void main(String[] args) {
26        launch(args);
27    }
28
29
30
31
32
33
34 D     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```

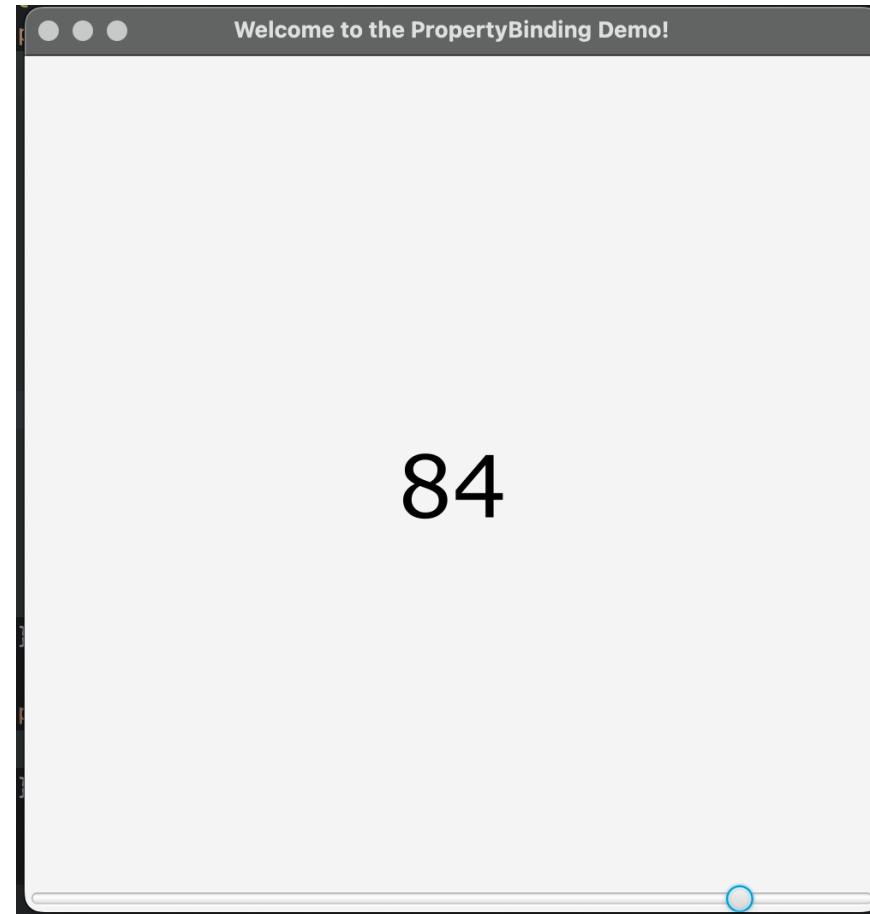
```
1 package com.COMP2013;
2 > import ...
3 
4 ▶ public class HelloApplication extends Application {
5     @Override
6     public void start(Stage stage) throws IOException {
7         stage.setTitle("Welcome to the PropertyBinding Demo!");
8         Slider slider = new Slider( v: 0, v1: 100, v2: 10);
9         Text text = new Text(String.valueOf((int)slider.getValue()));
10        text.setFont(Font.font( s: "Verdana", v: 50));
11        StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
12        slider.valueProperty().addListener(new ChangeListener<Number>() {
13            @Override
14            public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
15                text.setText(String.valueOf((int)slider.getValue()));
16            }
17        });
18        StackPane root = new StackPane();
19        root.getChildren().addAll(slider, text);
20        stage.setScene(new Scene(root, v: 500, v1: 500));
21        stage.show();
22    }
23 
24 ▶     public static void main(String[] args) {
25         launch(args);
26     }
27 }
```

```
1 package com.COMP2013;
2 > import ...
3
4 D public class HelloApplication extends Application {
5
6     @Override
7     public void start(Stage stage) throws IOException {
8         stage.setTitle("Welcome to the PropertyBinding Demo!");
9         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
10        Text text = new Text(String.valueOf((int)slider.getValue()));
11        text.setFont(Font.font(s: "Verdana", v: 50));
12        StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
13        slider.valueProperty().addListener(new ChangeListener<Number>() {
14            @Override
15            public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
16                text.setText(String.valueOf((int)slider.getValue()));
17            }
18        });
19        StackPane root = new StackPane();
20        root.getChildren().addAll(slider, text);
21        stage.setScene(new Scene(root, v: 500, v1: 500));
22        stage.show();
23    }
24
25    public static void main(String[] args) {
26        launch(args);
27    }
28
29
30
31
32
33
34 D     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```

```
1 package com.COMP2013;
2 > import ...
3
4 D public class HelloApplication extends Application {
5
6     @Override
7     public void start(Stage stage) throws IOException {
8         stage.setTitle("Welcome to the PropertyBinding Demo!");
9         Slider slider = new Slider(v: 0, v1: 100, v2: 10);
10        Text text = new Text(String.valueOf((int)slider.getValue()));
11        text.setFont(Font.font(s: "Verdana", v: 50));
12        StackPane.setAlignment(slider, Pos.BOTTOM_LEFT);
13
14         slider.valueProperty().addListener(new ChangeListener<Number>() {
15             @Override
16             public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {
17                 text.setText(String.valueOf((int)slider.getValue()));
18             }
19         });
20
21         StackPane root = new StackPane();
22         root.getChildren().addAll(slider, text);
23         stage.setScene(new Scene(root, v: 500, v1: 500));
24         stage.show();
25     }
26
27
28     public static void main(String[] args) {
29         launch(args);
30     }
31
32 }
33
34 D
35
36
37 }
```

</>

DEMO



The Node Class

</>

- Fundamental to JavaFX
- Used to represent controls, layouts, shapes, etc.
- Can apply effects (transform, translate etc.) to nodes

A screenshot of a JavaDoc interface showing the Node class. The top navigation bar includes links for OVERVIEW, MODULE, PACKAGE, CLASS (which is highlighted in orange), USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation bar, there are links for SUMMARY, NESTED, FIELD, CONSTR, and METHOD. A search bar is present with a magnifying glass icon and the word "Search".

Module javafx.graphics
Package javafx.scene

Class Node

java.lang.Object[↳]
javafx.scene.Node

All Implemented Interfaces:

Styleable, EventTarget

Direct Known Subclasses:

Camera, Canvas, ImageView, LightBase, MediaView, Parent, Shape, Shape3D, SubScene, SwingNode

A screenshot of a JavaDoc interface showing the Button class. The top navigation bar includes links for OVERVIEW, MODULE, PACKAGE, CLASS (which is highlighted in orange), USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation bar, there are links for SUMMARY, NESTED, FIELD, CONSTR, and METHOD. A search bar is present with a magnifying glass icon and the word "Search".

Module javafx.controls
Package javafx.scene.control

Class Button

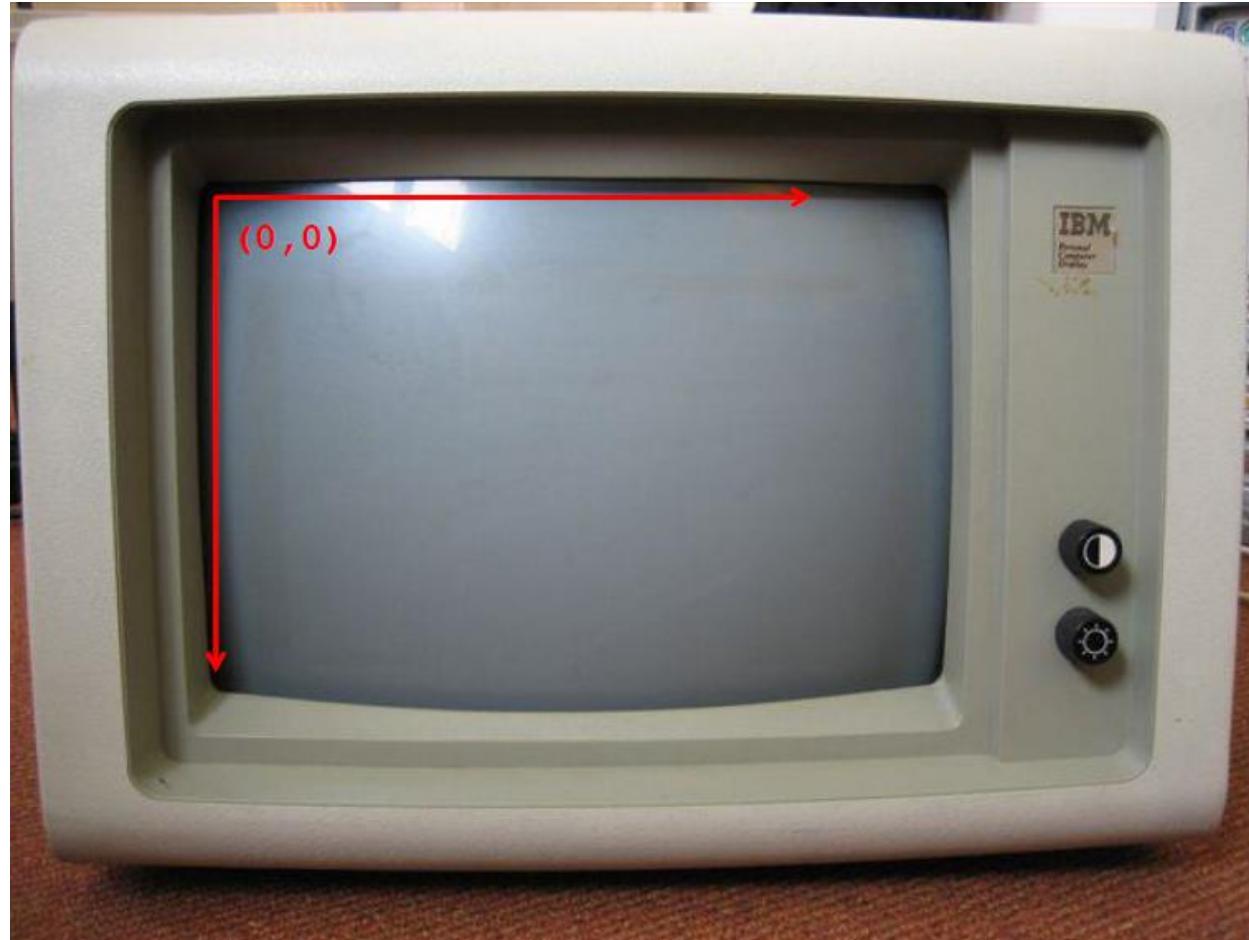
java.lang.Object[↳]
javafx.scene.Node
javafx.scene.Parent
javafx.scene.layout.Region
javafx.scene.control.Control
javafx.scene.control.Labeled
javafx.scene.control.ButtonBase
javafx.scene.control.Button

All Implemented Interfaces:

Styleable, EventTarget, Skinnable

Screen Coordinates

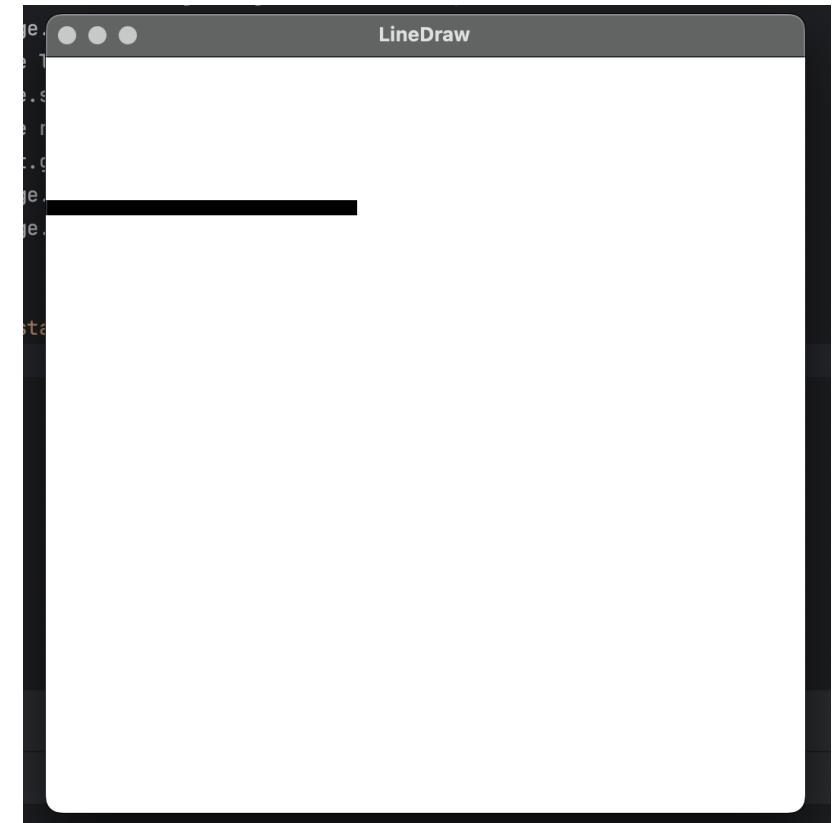
</>



Adding simple 2D Graphics

</>

```
1 package com.comp2013;
2 > import ...
3 ▶ public class LineApp extends Application {
4     @Override
5     public void start(Stage stage) throws IOException {
6         stage.setTitle("LineDraw");
7         Line line = new Line(v: 0, v1: 100, v2: 200, v3: 100);
8         line.setStrokeWidth(10);
9         Pane root = new Pane();
10        root.getChildren().add(line);
11        stage.setScene(new Scene(root, v: 500, v1: 500));
12        stage.show();
13    }
14
15    ▶ > public static void main(String[] args) { launch(args); }
16
17 }
```

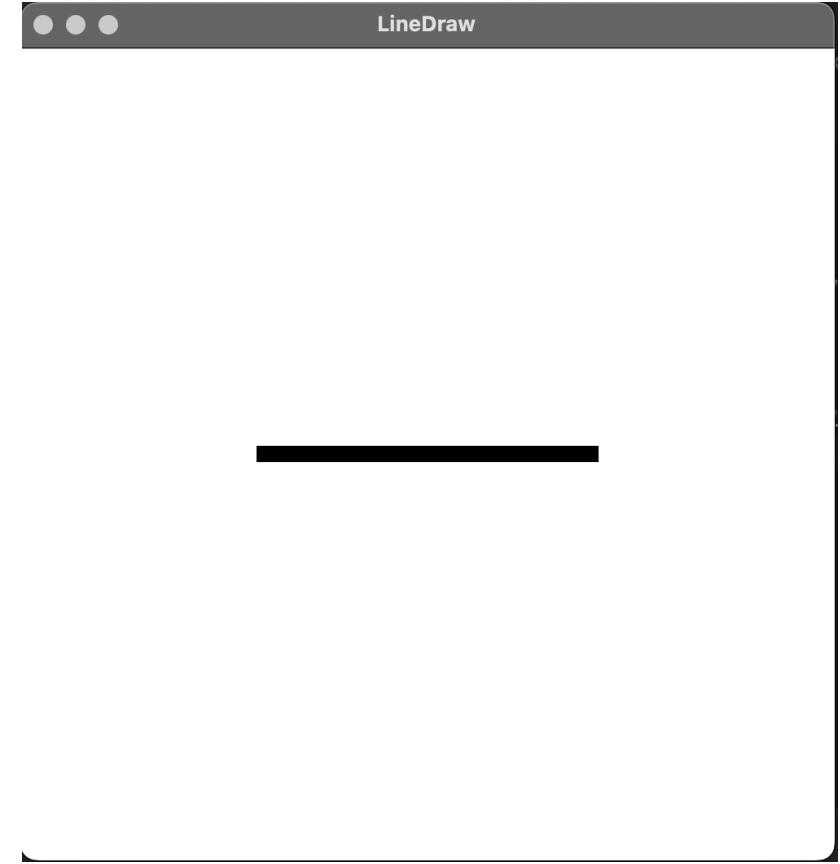


Adding simple 2D Graphics

</>

© LineApp.java ×

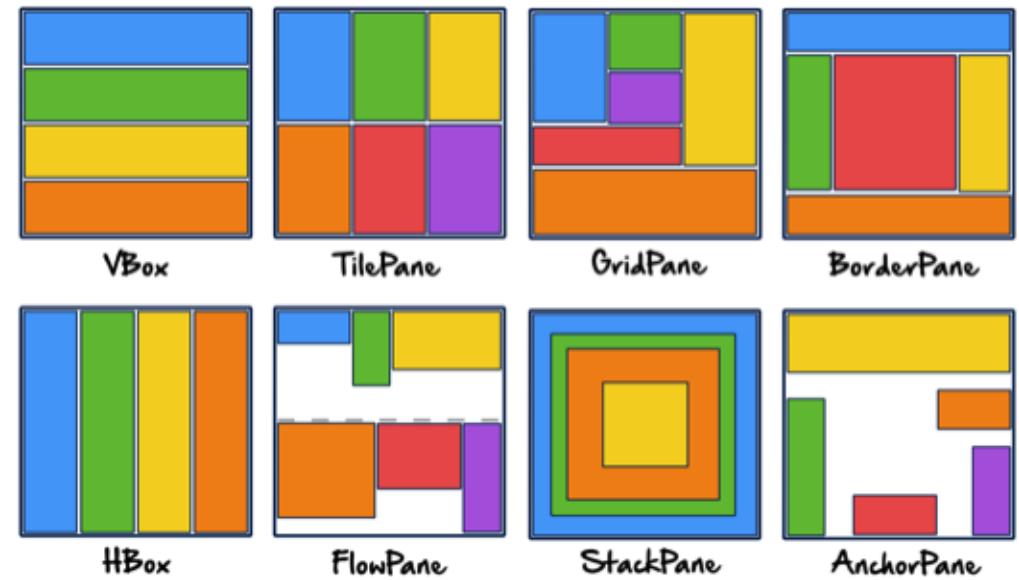
```
1 package com.comp2013;
2 > import ...
10 > public class LineApp extends Application {
11     @Override
12     public void start(Stage stage) throws IOException {
13         stage.setTitle("LineDraw");
14         Line line = new Line(v: 0, v1: 100, v2: 200, v3: 100);
15         line.setStrokeWidth(10);
16         Pane root = new StackPane();
17         root.getChildren().add(line);
18         stage.setScene(new Scene(root, v: 500, v1: 500));
19         stage.show();
20     }
21
22     public static void main(String[] args) { launch(args); }
23
24 }
```

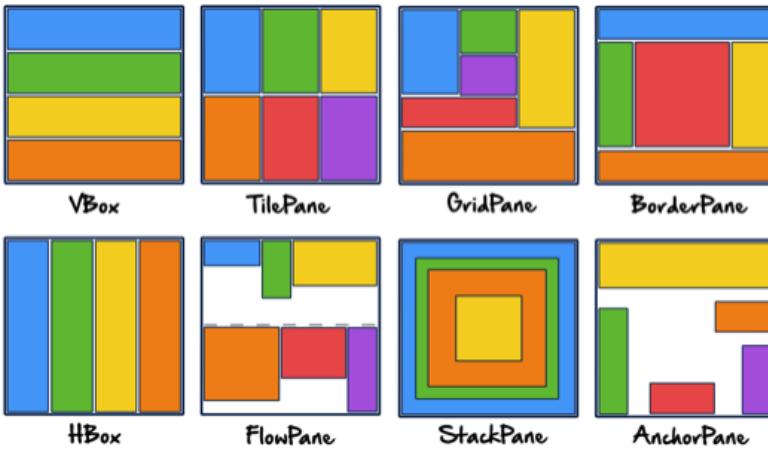


Built-in Layout Panes

</>

- **Vbox:** Provides an easy way for arranging a series of nodes in a single row
- **TilePane:** Places all of the nodes in a grid in which each cell, or tile, is the same size. Nodes can be laid out horizontally (in rows) or vertically (in columns)
- **GridPane:** Allows to create a flexible grid of rows and columns in which to lay out nodes; nodes can be placed in any cell in the grid and can span cells as needed
- **BorderPane:** Provides five regions in which to place nodes: top, bottom, left, right, and centre
- **HBox:** Provides an easy way for arranging a series of nodes in a single column
- **FlowPane:** Nodes are laid out consecutively and wrap at the boundary set for the pane; nodes can flow vertically (in columns) or horizontally (in rows)
- **StackPane:** Places all of the nodes within a single stack with each new node added on top of the previous
- **AnchorPane:** Allows to anchor nodes to the top, bottom, left side, right side, or centre of the pane; as the window is resized, the nodes maintain their position relative to their anchor point.





</>

```

resources
JavaFXPaneDemo
one two three
four
five five five

FlowPane root=new FlowPane();
HBox hbox=new HBox(btn1,btn2);
VBox vbox=new VBox();
vbox.getChildren().add(btn3);
vbox.getChildren().add(btn4);
root.getChildren().addAll(hbox,vbox,btn5);
//root.getChildren().addAll(btn1,btn2,btn3,btn4,btn5);
stage.setScene(new Scene(root, 500, 500));
stage.show();

}

public static void main(String[] args) {
    launch(args);
}
}

n/java ...

```

Linking JavaFX and Swing

Introduction

```
1 package com.comp2013;
2
3 > import ...
4
5
6 ▶ public class HelloApplication extends Application {
7     @Override
8         public void start(Stage stage) throws IOException {
9             stage.setTitle("Hello Java FX");
10            Button button1 = new Button(s: "Close App");
11            button1.setOnAction(new EventHandler<ActionEvent>() {
12                @Override
13                    public void handle(ActionEvent actionEvent) {
14                        System.out.println("Bye Bye!");
15                        System.exit(status: 0);
16                    }
17            });
18            StackPane root = new StackPane();
19            root.getChildren().add(button1);
20            stage.setScene(new Scene(root, v: 600, v1: 600));
21            stage.show();
22        }
23
24    public static void main(String[] args) {
25        launch();
26    }
27
28 }
```

```
1 package com.COMP2013;
2 > import ...
3
4 ▶ public class HelloWorld SwingGUIApp extends JFrame {
5     1 usage
6         public HelloWorld SwingGUIApp(){
7             initUI();
8         }
9
10    1 usage
11        private void initUI(){
12            setSize(width: 600, height: 600);
13            setTitle("My first GUI in Java Swing. Hello world!");
14            setLocationRelativeTo(null);
15            setDefaultCloseOperation(EXIT_ON_CLOSE);
16            JPanel pannel = new JPanel(new GridLayout());
17            JButton button1 = new JButton(text: "Exit program");
18            button1.addActionListener(new ActionListener() {
19                @Override
20                    public void actionPerformed(ActionEvent e) {
21                        System.out.println("Now we will exit the program");
22                        System.exit(status: 0);
23                    }
24            );
25            pannel.add(button1);
26            this.add(pannel);
27
28 }
```

Linking Swing to JavaFX

</>



some final remarks ...



The University of
Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Lecture 03B

Maintainable GUI Development (1/2)

Introduction to GUI Programming in Java



Horia A. Maior and Marjahan Begum

COMP2013: Developing Maintainable Software
Week 4 – 1:00pm Thursday – 19 October 2023



valid for 65 minutes from 12:55pm
generated 2023-10-10 03:14

</>



COMP2013: Developing Maintainable Software
Week 4 – 1:00pm Thursday – 19 October 2023

Lecture 03B

Maintainable GUI Development (1/2)

GUI Development in Practice

Horia A. Maior and Marjahan Begum



Topics for this Week

</>

- Lecture 03A
 - Introduction to GUI programming in Java
 - GUIs; Swing vs JavaFX; JavaFX foundations
- Lab 03
 - Practice JavaFX and Swing GUI development (phone app)
- Lecture 03B
 - GUI development in practice
 - Lab reflection;
 - MVC Pattern
 - FXML;
 - multiple windows

COMP2013: Developing Maintainable Software
Week 4 – 1:00pm Thursday – 19 October 2023



</>



valid for 65 minutes from 12:55pm
generated 2023-10-10 03:14

lab reflection

LAB 3: SIMPLE GUI DEVELOPMENT WITH JAVAFX and SWING

Aims:

- Ensure that JavaFX and SWING work on your computer
- Gain some GUI development experience by re-implementing Lecture 03A's examples
- Gain some GUI development experience by building a simple phone app in SWING and JAVAFX.

Please do the exercise(s) that you feel are appropriate for you.

BEGINNER LEVEL: CREATING A JAVAFX PROJECT

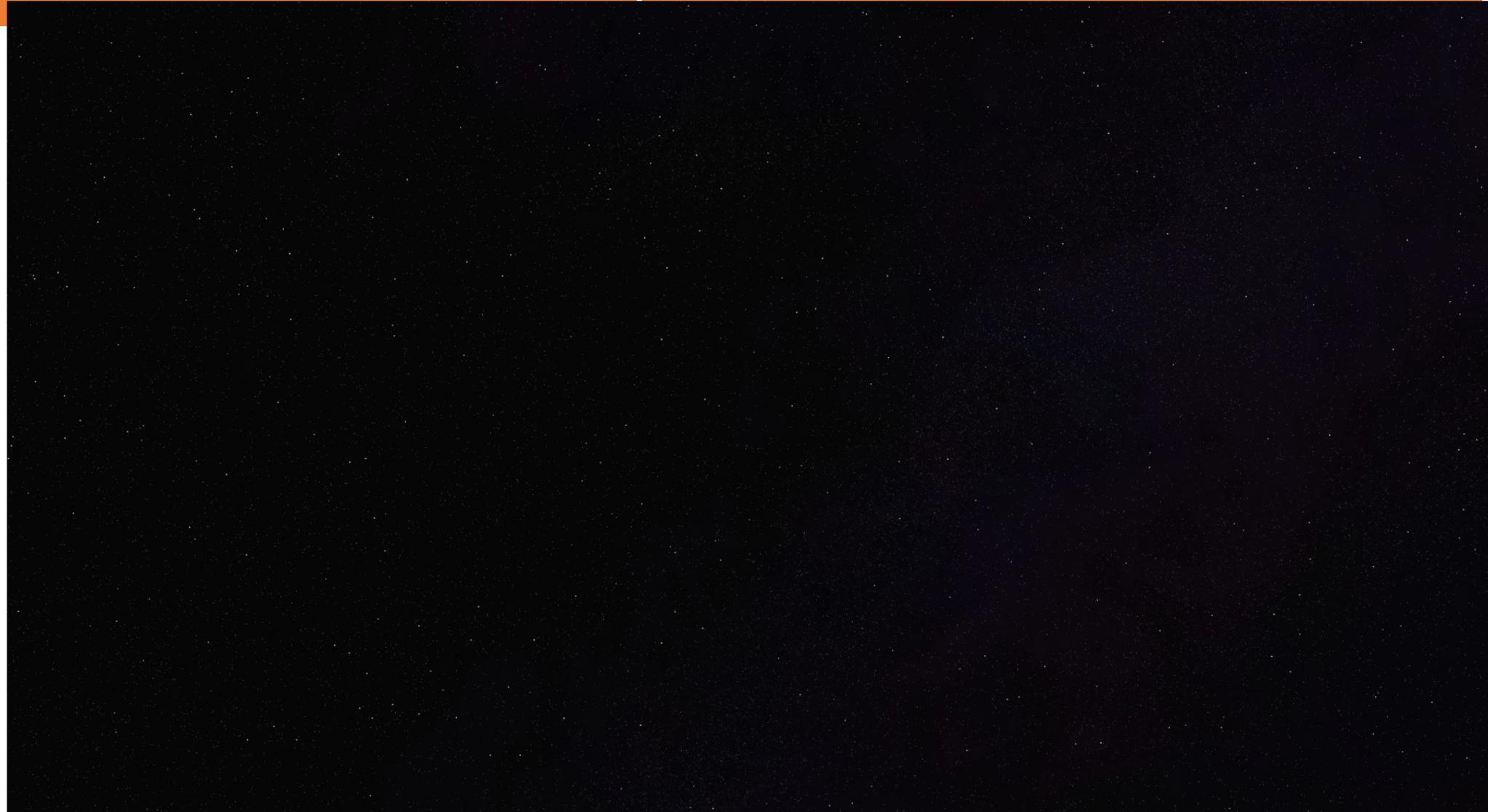
To ensure that JavaFX works on your computer, set up some test projects in IntelliJ. Start by doing it manually (as I did in the lecture, using a plain Java project) and then by using the JavaFX project option, which will configure some build script to set up the project and dependencies automatically.

Part 1: RE-IMPLEMENTING and MODIFYING THE LECTURE EXAMPLE in JAVAFX and SWING

Re-implement the HelloJavaFXWorld example presented in Lecture 03A. Then do some modifications to it (e.g. adding components or using different layout panes) and check out the consequences.

Lab Reflection JavaFX Setup

</>



Your main task for the lab is to build a simple telephone application. You can find the spec below.

Specification: You need to build a JavaFX GUI-based app with the following features:

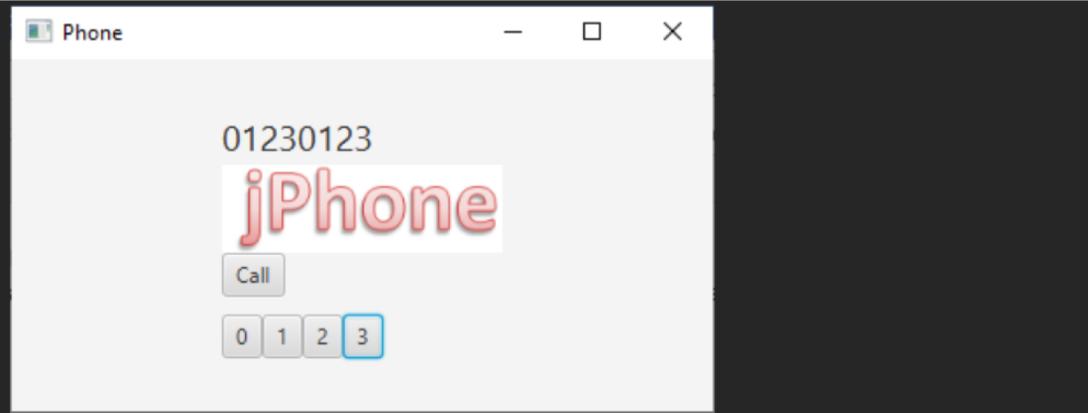
1. Number buttons (0 to 9) to enter the phone number {**Tip: A GridPane might be useful here**}
2. A display to view the current number
3. A Call button. This will obviously not actually make a call, but should pass the number to a *makeCall()* method, which implements an *Phone* interface. This class should represent a dummy telephone device, and the method should wait for a random duration (in seconds) and then return the duration of the "call".

```
//Things which can make or fake telephone calls implement this.  
public interface Phone {  
    public int makeCall(String number);  
}
```

- o Note how, by implementing an interface, we can switch out a "dummy" or "real" phone application easily.
- o To wait for some time you could do something like this (note though, that this example snippet does not wait for a random time, and that this might not be the best way to deal with this requirement):

```
duration=3000; // could make this random  
try {  
    Thread.sleep(duration); // note how this hangs the interface! how can we fix that?  
} catch (InterruptedException e) {  
    e.printStackTrace();  
} // faking phone call time.
```





5. Maybe think on how you could make this look pretty 😊

Things to bear in mind:

- You need to handle telephone numbers starting with a 0, so think about your data type!

Part 3: REWRITE THE TELEPHONE APPLICATION in SWING

Now that you have managed to finalise your design in JavaFX, we want you to build the same program but using Swing. This is related to legacy code; a lot of projects out there are built in Swing, so learning and understanding will be highly beneficial.

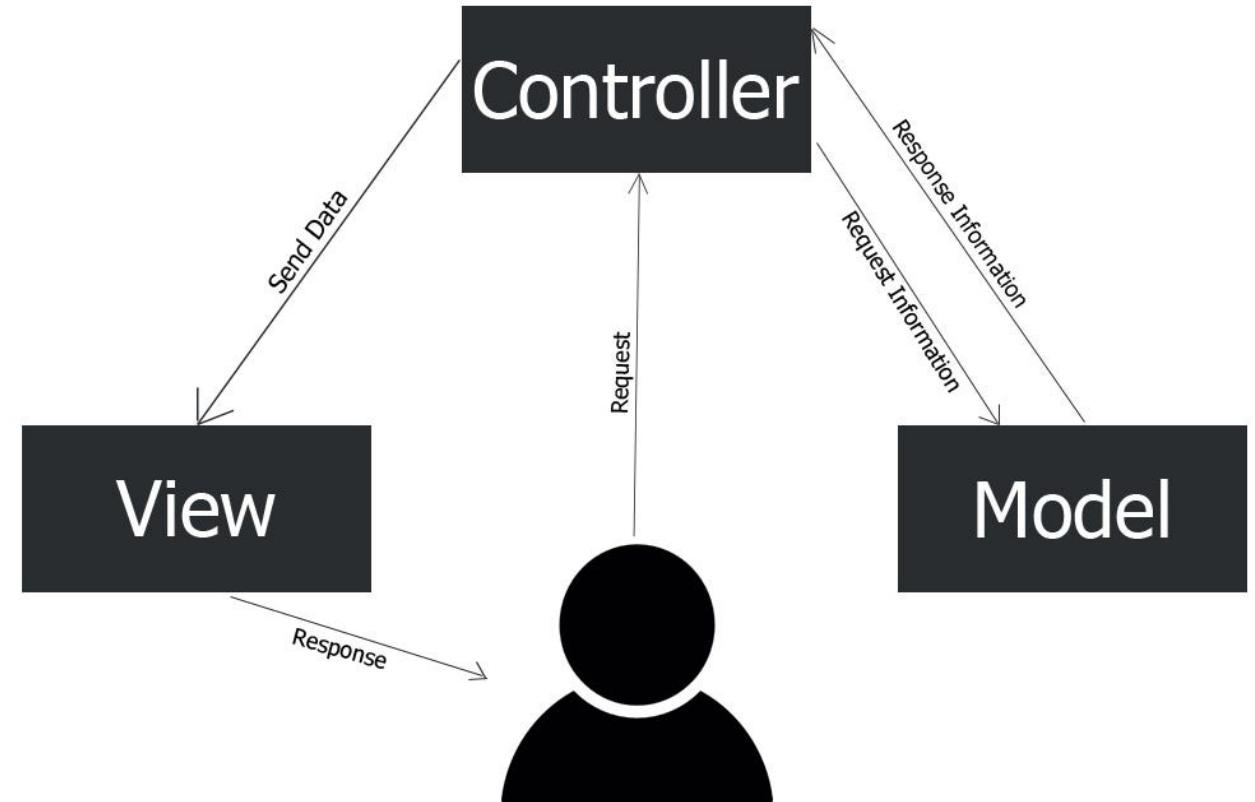
- Start from the Hello World Example in the Monday Lecture, and build on from there. This will require some individual research into how to build different parts.
- For those of you not so familiar with Swing, here are some excellent tutorials to follow in [Java Swing](#)

mvc pattern and FXML

Model-View-Controller

Model-View-Controller

- View tells controller what happened
- Controllers inform the model what to do
- Once the job is finished the model broadcasts notifications; it is not aware of any specific view/controller
- Views catches relevant notifications and updates



https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5

MVC Applied in Java

</>

DEMO

The screenshot shows an IDE interface with a dark theme. On the left is the Project Explorer, which lists the project structure for 'JavaFXSetup'. A red box highlights the 'src' folder, which contains 'main' and 'java'. The 'java' folder is expanded, showing packages 'com.COMP2013' and 'example', and files 'Bike', 'BikeApp', 'BikeController', 'BikeModel', and 'module-info.java'. Below 'java' is a 'resources' folder containing 'com/COMP2013/hello-view.fxml' and 'example'. The right side of the interface shows the code editor with 'BikeController.java' open. The code defines a 'Bike' class with private attributes 'type' and 'color', and methods for getting and setting 'type'. The code editor also shows 'hello-view.fxml' and 'BikeModel.java' in the tabs.

```
package com.COMP2013;

public class Bike {
    private String type;
    private String color;

    public Bike(String type, String color){
        this.type=type;
        this.color=color;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
}
```

See <https://stackoverflow.com/questions/36868391/using-javafx-controller-without-fxml> for more explanations

MVC Applied in Java

</>

DEMO

```
© BikeApp.java ×

Runnable class package com.COMP2013;
import ...

8

9 D public class BikeApp extends Application {
10
11 ⓘ @Override
12
13 public void start(Stage stage) throws IOException {
14     FXMLLoader fxmlLoader = new FXMLLoader(BikeApp.class.getResource(name: "hello-view.fxml"));
15     Scene scene = new Scene(fxmlLoader.load(), v: 500, v1: 500);
16     stage.setTitle("AddBikesApp!");
17     stage.setScene(scene);
18     stage.show();
19 }
20
21 D public static void main(String[] args) {
22     launch();
23 }
```



MVC A

© BikeApp.java © BikeController.java </> hello-view.fxml ×

</>

DEMO

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?import javafx.geometry.Insets?>
3  <?import javafx.scene.control.Label?>
4  <?import javafx.scene.layout.VBox?>
5  <?import javafx.scene.control.Button?>
6  <?import javafx.scene.control.ListView?>
7  <?import javafx.scene.control.TextField?>
8
9  <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
10   fx:controller="com.COMP2013.BikeController">
11  <padding>
12    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
13  </padding>
14  <Label text="Check out the list of my bike objects below:"/>
15  <ListView fx:id="bikeListView" />
16  <Label text="Bike Name"></Label>
17  <TextField fx:id="bikeNameField"/>
18  <Label text="Bike Color"></Label>
19  <TextField fx:id="bikeColorField"/>
20  <Button text="Add Bike" onAction="#onAddBikeBtnClick" />
21  </VBox>
```



MVC A

© BikeApp.java © BikeController.java </> hello-view.fxml ×

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?import javafx.geometry.Insets?>
3  <?import javafx.scene.control.Label?>
4  <?import javafx.scene.layout.VBox?>
5  <?import javafx.scene.control.Button?>
6  <?import javafx.scene.control.ListView?>
7  <?import javafx.scene.control.TextField?>
8
9  <VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
10   fx:controller="com.COMP2013.BikeController">
11  <padding>
12    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
13  </padding>
14  <Label text="Check out the list of my bike objects below:"/>
15  <ListView fx:id="bikeListView" />
16  <Label text="Bike Name"></Label>
17  <TextField fx:id="bikeNameField"/>
18  <Label text="Bike Color"></Label>
19  <TextField fx:id="bikeColorField"/>
20  <Button text="Add Bike" onAction="#onAddBikeBtnClick" />
21  </VBox>
```

</>

DEMO

MVC Applied in Java

© BikeApp.java © BikeController.java × </> hello-view.fxml

</>

DEMO

```
1 package com.COMP2013;
2 > import ...
3
4 public class BikeController {
5
6     @FXML
7     private ListView<Bike> bikeListView;
8
9     @FXML
10    private TextField bikeNameField;
11
12    @FXML
13    private TextField bikeColorField;
14
15    private BikeModel model;
16
17    public void initialize() {
18        // Set up the ListView to display the list of bikes
19        model = new BikeModel();
20        ObservableList<Bike> bikes = model.getBikes();
21        bikeListView.setItems(bikes);
22    }
23
24    @FXML
25    private void onAddBikeBtnClick() {
26        String name = bikeNameField.getText();
27        String color = bikeColorField.getText();
28        Bike newBike = new Bike(name, color);
29        model.addBike(newBike);
30    }
31 }
```

MVC A

© BikeApp.java

© BikeController.java

© BikeModel.java × </> hello-view.fxml

</>

DEMO

```
1 package com.COMP2013;
2 > import ...
4
5 public class BikeModel {
6     5 usages
7         private ObservableList<Bike> bikes = FXCollections.observableArrayList();
8         public BikeModel() {
9             // Initialize the model with some sample data
10            bikes.add(new Bike(type: "Mountain Bike", color: "Blue"));
11            bikes.add(new Bike(type: "Road Bike", color: "Red"));
12            bikes.add(new Bike(type: "City Bike", color: "Green"));
13        }
14        1 usage
15        > public ObservableList<Bike> getBikes() { return bikes; }
16
17        1 usage
18        > public void addBike(Bike bike) { bikes.add(bike); }
19
20    }
21
22 }
```



</>

© BikeApp.java © BikeController.java © BikeModel.java © Bike.java ×

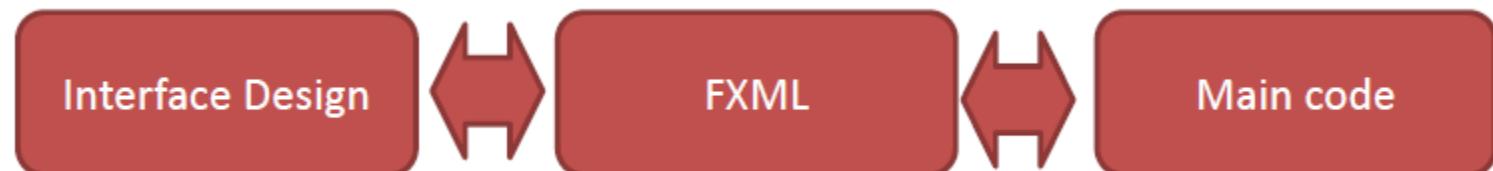
```
1 package com.COMP2013;
2
3 public class Bike {
4     3 usages
5     private String type;
6     3 usages
7     private String color;
8
9     7 usages
10    public Bike(String type, String color){
11        this.type=type;
12        this.color=color;
13    }
14
15    no usages
16    public String getType() { return type; }
17    no usages
18    public void setType(String type) { this.type = type; }
19    no usages
20    public String getColor() { return color; }
21    no usages
22    public void setColor(String color) { this.color = color; }
23
24 }
25 |
```

COMP2013

FXML

Introduction

- So far we have seen how we can build simple GUIs in code
- BUT if the GUI is going to get complicated JavaFX supports describing the layout using FXML (FX Markup Language)
- Supports the idea of separating 'Design' and 'Functionality'
 - Languages like C# in Visual Studio also support this separation of concerns

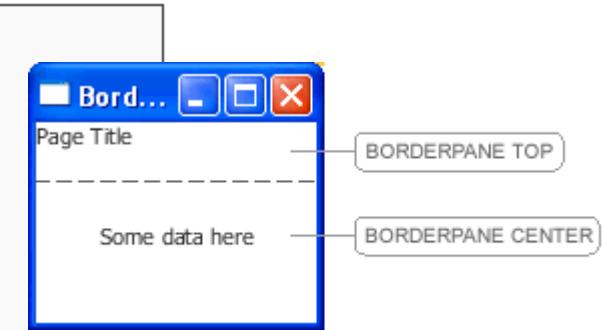


JavaFX vs FXML

</>

Example 1-1 Java Code for a User Interface

```
BorderPane border = new BorderPane();
Label toppanetext = new Label("Page Title");
border.setTop(toppanetext);
Label centerpanetext = new Label ("Some data here");
border.setCenter(centerpanetext);
```



Example 1-2 FXML Markup for a User Interface

```
<BorderPane>
    <top>
        <Label text="Page Title"/>
    </top>
    <center>
        <Label text="Some data here"/>
    </center>
</BorderPane>
```

Simpler, less code, and a more
intuitive organisation

Creating FXML Project

New Project

Name: demo

Location: ~/Software Development/COMP2013

Project will be created in: ~/Software Development/COMP2013/demo

Create Git repository

Build system: Maven Gradle

Group: com.example

Artifact: demo

JDK: 20 Oracle OpenJDK version 20.0.2

Search:

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Create Quarkus applications

Micronaut

Ktor

Kotlin Multiplatform

Compose for Desktop

HTML

React

Express

Angular CLI

IDE Plugin

Android

Vue.js

Vite

Play 2

Flask

FastAPI

Django

Composer Package Project

?

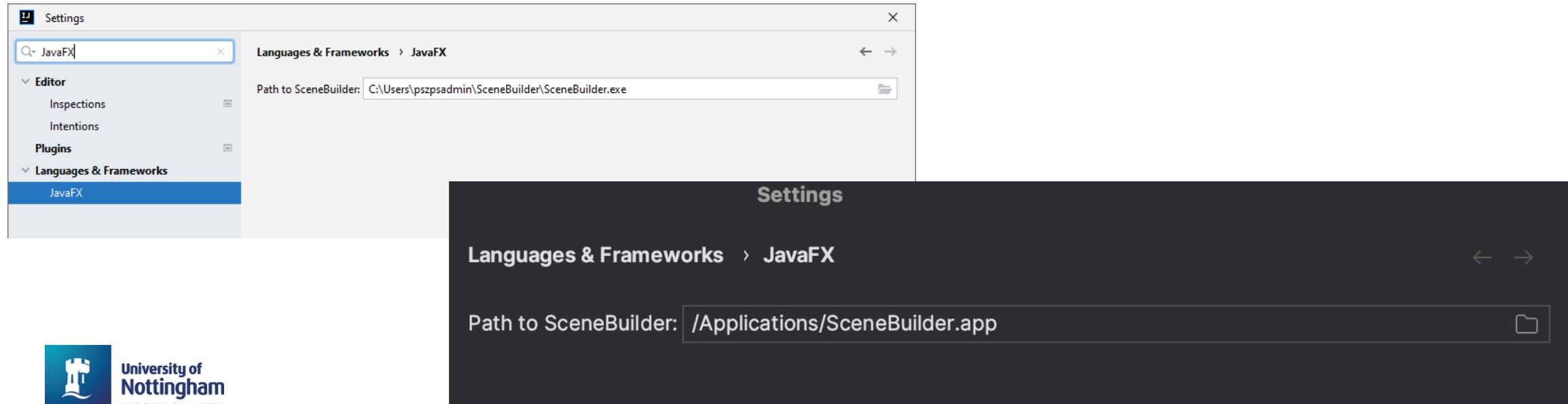
Cancel

Next

Preparing FXML Editor

</>

- Scene Builder:
 - A design tool for created FXML files graphically
 - Download from <https://gluonhq.com/products/scene-builder/> and install
 - You need to tell the IDE where to find the Gluon Scene Builder executable
 - <https://www.jetbrains.com/help/idea/opening-fxml-files-in-javafx-scene-builder.html#open-in-scene-builder>



Open FXML Editor

</>

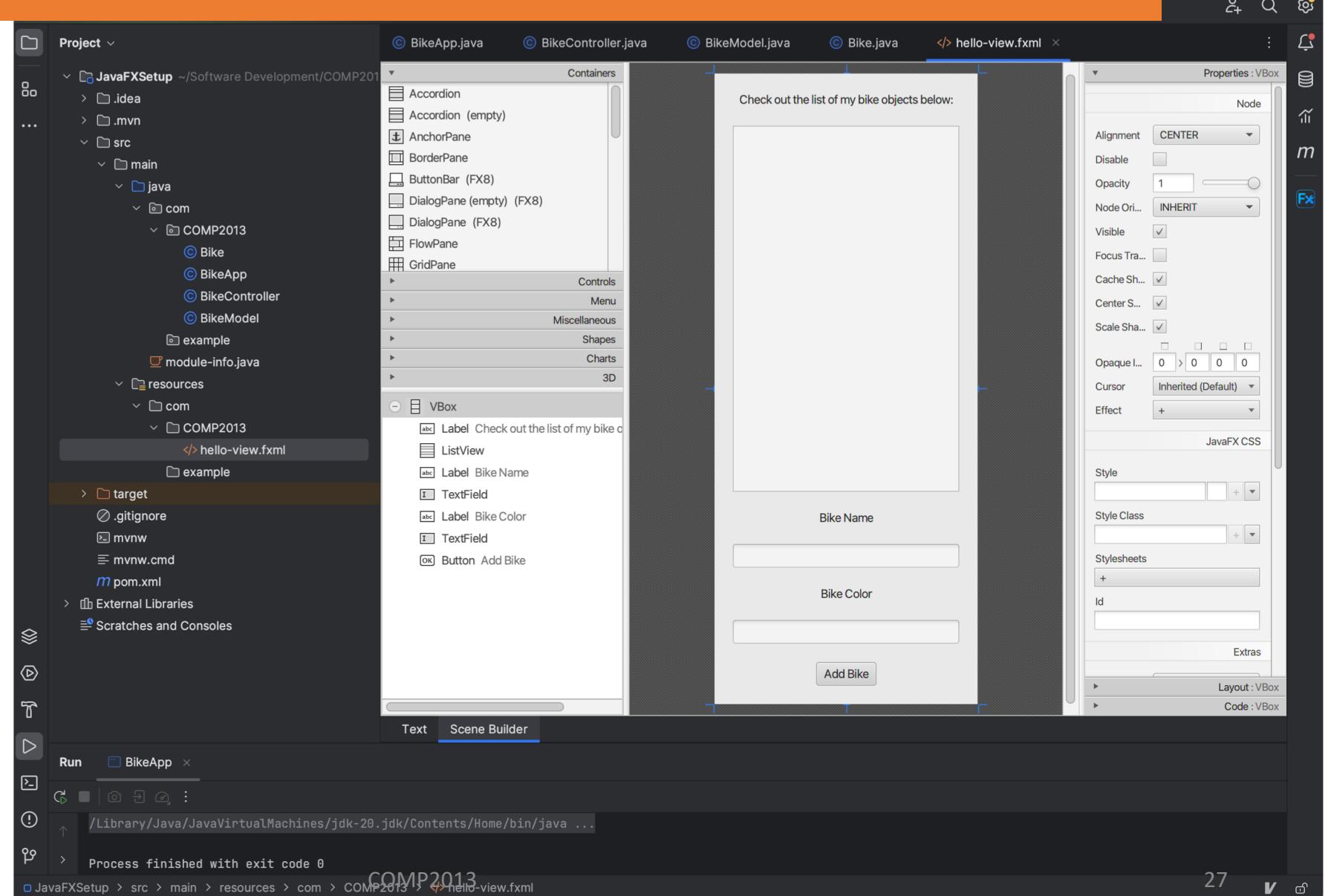
The screenshot shows the JavaFXSetup IDE interface. The left sidebar displays the project structure for 'JavaFXSetup' under '~/Software Development/COMP201'. The 'src' folder contains 'main', which has 'java' and 'com'. The 'com' folder contains 'COMP2013' with classes 'Bike', 'BikeApp', 'BikeController', 'BikeModel', and an 'example' folder containing 'module-info.java'. The 'resources' folder contains 'com/COMP2013/hello-view.fxml'. The right pane shows the content of 'hello-view.fxml' with code highlighting for JavaFX elements like `<VBox>`, `<Label>`, and `<TextField>`. The status bar at the bottom shows tabs for 'Text' and 'Scene Builder', with 'Text' currently selected. A red box highlights the 'Text' tab.

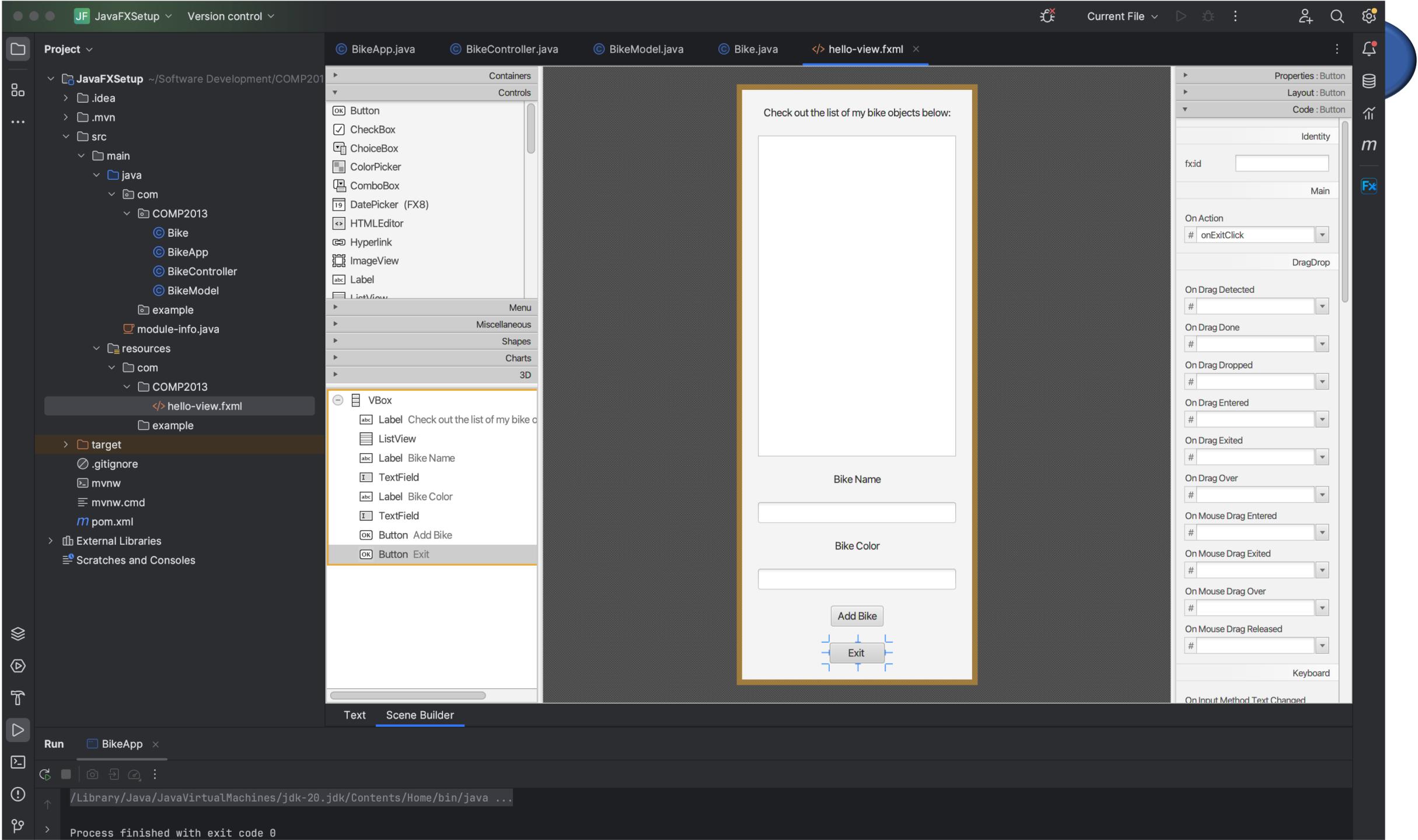
```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.control.TextField?>

<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
       fx:controller="com.COMP2013.BikeController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>
    <Label text="Check out the list of my bike objects below:"/>
    <ListView fx:id="bikeListView" />
    <Label text="Bike Name"></Label>
    <TextField fx:id="bikeNameField"/>
    <Label text="Bike Color"></Label>
    <TextField fx:id="bikeColorField"/>
    <Button text="Add Bike" onAction="#onAddBikeBtnClick" />
</VBox>
```

</>

Open FXML Editor





The screenshot shows the JavaFXSetup IDE interface. The left sidebar displays the project structure for 'JavaFXSetup' under '~/Software Development/COMP201'. The 'src' folder contains 'main', 'java', 'com', 'COMP2013', 'resources', and 'target'. 'COMP2013' contains 'Bike', 'BikeApp', 'BikeController', 'BikeModel', 'example', and 'module-info.java'. 'resources' contains 'com' and 'COMP2013', which in turn contains 'hello-view.fxml'. The right pane shows the content of 'hello-view.fxml' with the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/11.0.14-internal" fx:controller="com.COMP2013.Bik
<padding>
    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />
</padding>
<Label text="Check out the list of my bike objects below:" />
<ListView fx:id="bikeListView" />
<Label text="Bike Name" />
<TextField fx:id="bikeNameField" />
<Label text="Bike Color" />
<TextField fx:id="bikeColorField" />
<Button onAction="#onAddBikeBtnClick" text="Add Bike" />
<Button mnemonicParsing="false" onAction="#onExitClick" prefHeight="26.0" prefWidth="69.0" text="Exit " />
```

A red box highlights the last two lines of the code, which define a Button with mnemonic parsing set to false and an exit action.

The bottom status bar shows the path 'VBox > Button', tabs 'Text' and 'Scene Builder', and the message 'Process finished with exit code 0'.

Why use FXML?

</>



- Why use FXML? What do people think are the benefits?
 - UI designers might not be programmers
 - The designers can use external software (such as Scene Builder) to design the look of the UI whilst the programmers can build the functionality
 - FXML glues the two aspects together
 - Building GUIs visually rather than programmatically makes intuitive sense
 - Event handling is simplified
 - We can test/fix application logic without touching the GUI design or vice versa

JavaFX advanced topics

Multiple Windows

Multiple Windows

- Simple approach

```
m pom.xml (JavaFXMulti-Scenes)    </> hello-view.fxml    ○ HelloController.java    ○ HelloApplication.java ×
17  ▷  public class HelloApplication extends Application {
18      3 usages
19
20
21  ①@  private Scene scene1, scene2;
22
23
24
25
26
27  ①  @Override
28  public void start(Stage primaryStage) {
29      primaryStage.setTitle("Java FX Multiple Scenes");
30      Label label1= new Label(s: "Primary View");
31      Button button1= new Button(s: "Switch to Secondary View");
32      button1.setOnAction(new EventHandler<ActionEvent>() {
33          @Override
34          public void handle(ActionEvent actionEvent) {
35              primaryStage.setScene(scene2);
36          }
37      });
38
39  ①  VBox layout1 = new VBox(v: 20);
40  layout1.setAlignment(Pos.CENTER);
41  layout1.getChildren().addAll(label1, button1);
42  scene1= new Scene(layout1, v: 300, v1: 400);
43  Label label2= new Label(s: "Secondary View");
44  Button button2= new Button(s: "Switch to Primary View");
45  button2.setOnAction(new EventHandler<ActionEvent>() {
46      @Override
47      public void handle(ActionEvent actionEvent) {
48          primaryStage.setScene(scene1);
49      }
50  });
51  ①  VBox layout2= new VBox(v: 20);
52  layout2.setAlignment(Pos.CENTER);
53  layout2.getChildren().addAll(label2, button2);
54  scene2= new Scene(layout2, v: 400, v1: 300);
55  primaryStage.setScene(scene1);
56  primaryStage.show();
57
58
59
60
61  ▷  public static void main(String[] args) {
62      launch(args);
63  }
64 }
```

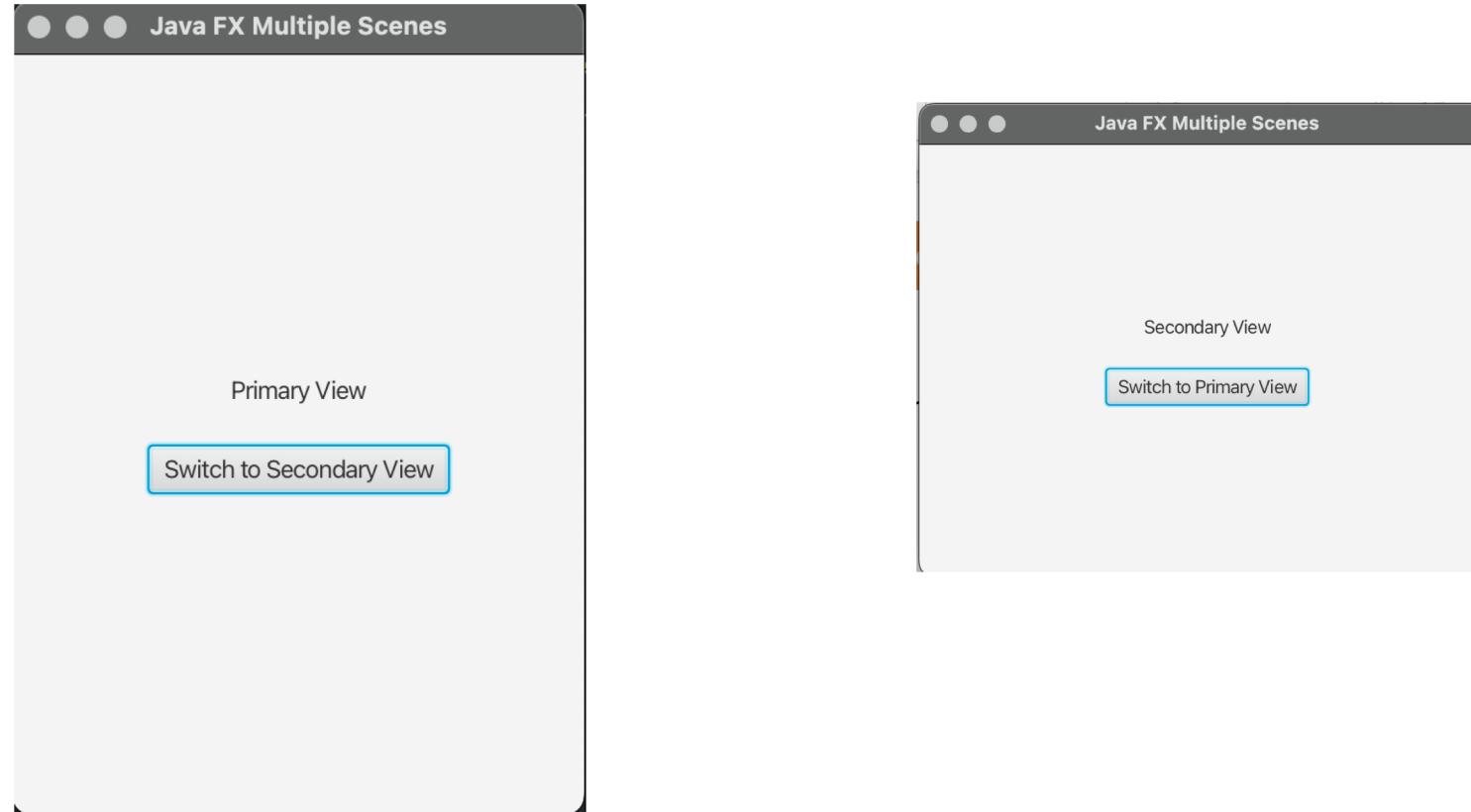
</>

DEMO

Multiple Windows

</>

- The result



Multiple Windows

</>

© MultiStageDemoApp.java × © PrimaryController.java </> primary.fxml © SecondaryController.java </> secondary.fxml

```
1 package com.example.COMP2013;
2
3 > import ...
10
11 > public class MultiStageDemoApp extends Application {
12     private static Scene scene;
13
14     @Override
15 @
16     public void start(Stage stage) throws IOException {
17         scene = new Scene(loadFXML("primary"));
18         stage.setScene(scene);
19         stage.show();
20     }
21
22     static void setRoot(String fxml) throws IOException {
23         scene.setRoot(loadFXML(fxml));
24     }
25
26     private static Parent loadFXML(String fxml) throws IOException {
27         //System.out.println(getClass().getResource(fxml + ".fxml"));
28         FXMLLoader fxmlLoader = new FXMLLoader(MultiStageDemoApp.class.getResource(name: fxml + ".fxml"));
29         return fxmlLoader.load();
30     }
31 >     public static void main(String[] args) { launch(); }
32
33 }
```

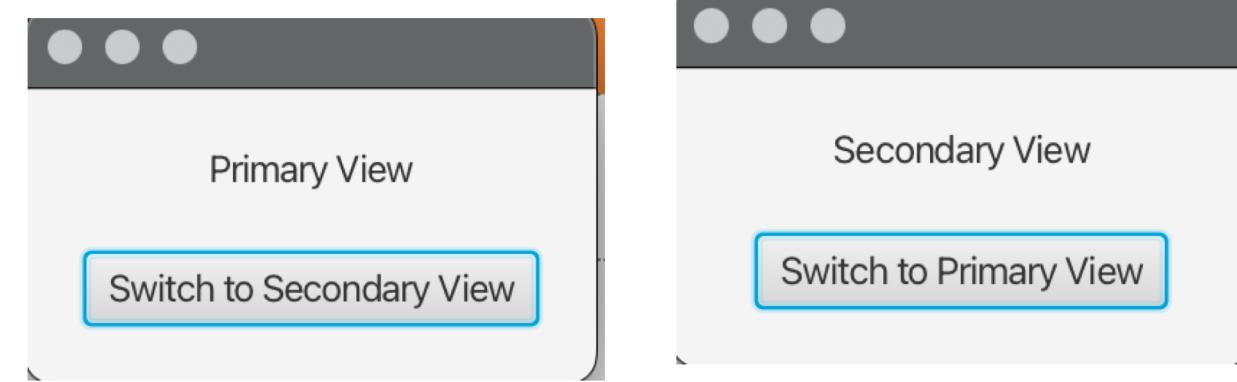
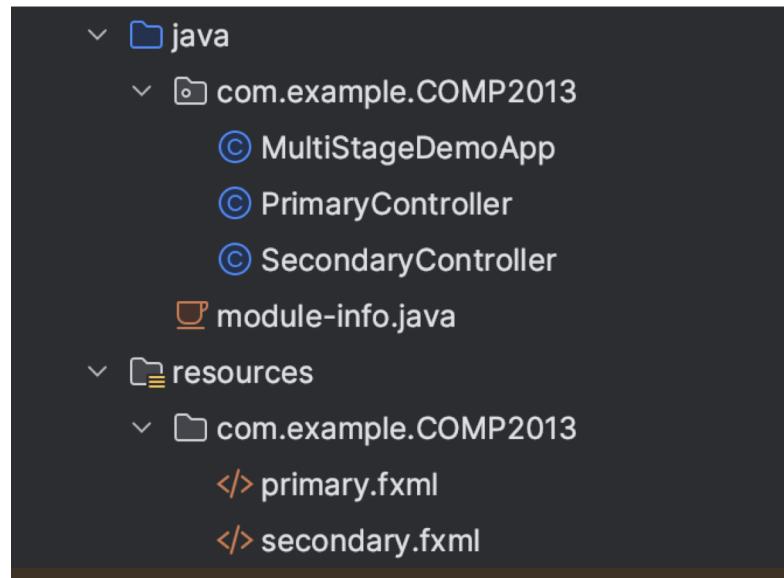
```
1 package com.example.COMP2013; ~/Software Development/COMP2013/M  
2  
3 import javafx.fxml.FXML;  
4  
5 import java.io.IOException;  
6  
7 public class PrimaryController {  
8  
9     @FXML  
10    private void switchToSecondary() throws IOException {  
11        MultiStageDemoApp.setRoot("secondary");  
12    }  
13 }  
14 |
```

```
1 package com.example.COMP2013;  
2  
3 > import ...  
4  
5 public class SecondaryController {  
6     no usages  
7     @FXML  
8     private Label welcomeText;  
9  
10    @FXML  
11    private void switchToPrimary() throws IOException {  
12        MultiStageDemoApp.setRoot("primary");  
13    }  
14 }  
15 }  
16 }
```

Multiple Windows

</>

- The result



<https://stackoverflow.com/questions/10134856/javafx-2-0-how-to-application-getparameters-in-a-controller-java-file/10136403#10136403>

Some final remarks ...



Lecture 4 - Object Oriented Analysis and Design with UML

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior



Register your attendance

COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



Overview

- Why UML?
- Introduction to 2 case studies
- Use case diagrams
- Use case specifications
- Activity diagrams
- Sequence diagrams
- State machine diagrams
- State machine
- Review class diagrams

COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



Topics for this Week

- Lecture 04A:
 - OO Analysis and Design (OOA/D) with UML
- Lecture 04B:
 - Build Tools (Maven and Gradle)
 - Introduction the Formative Group project
 - Preview of 2022/2023 coursework
- Lab 04:
 - Virtual UML Speed Refactor/Extend Challenge for **Group Project groups**

COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



Unified Modelling Language

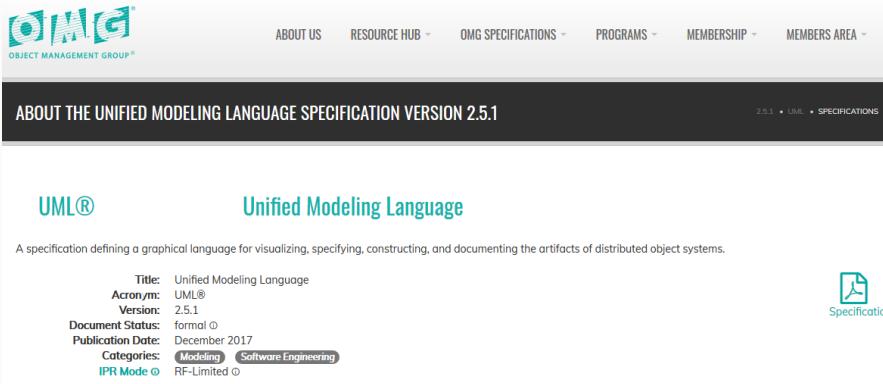


UML: An Overview

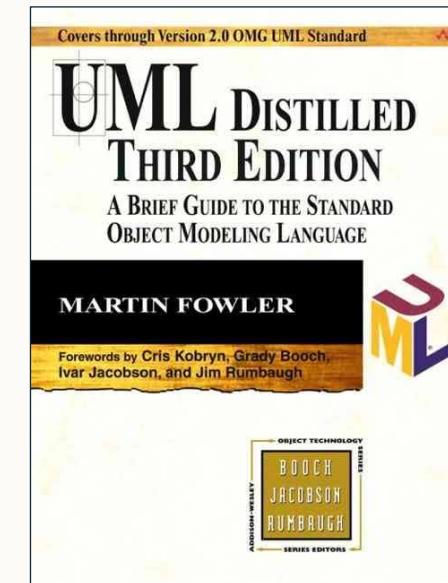
- UML: "A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems."

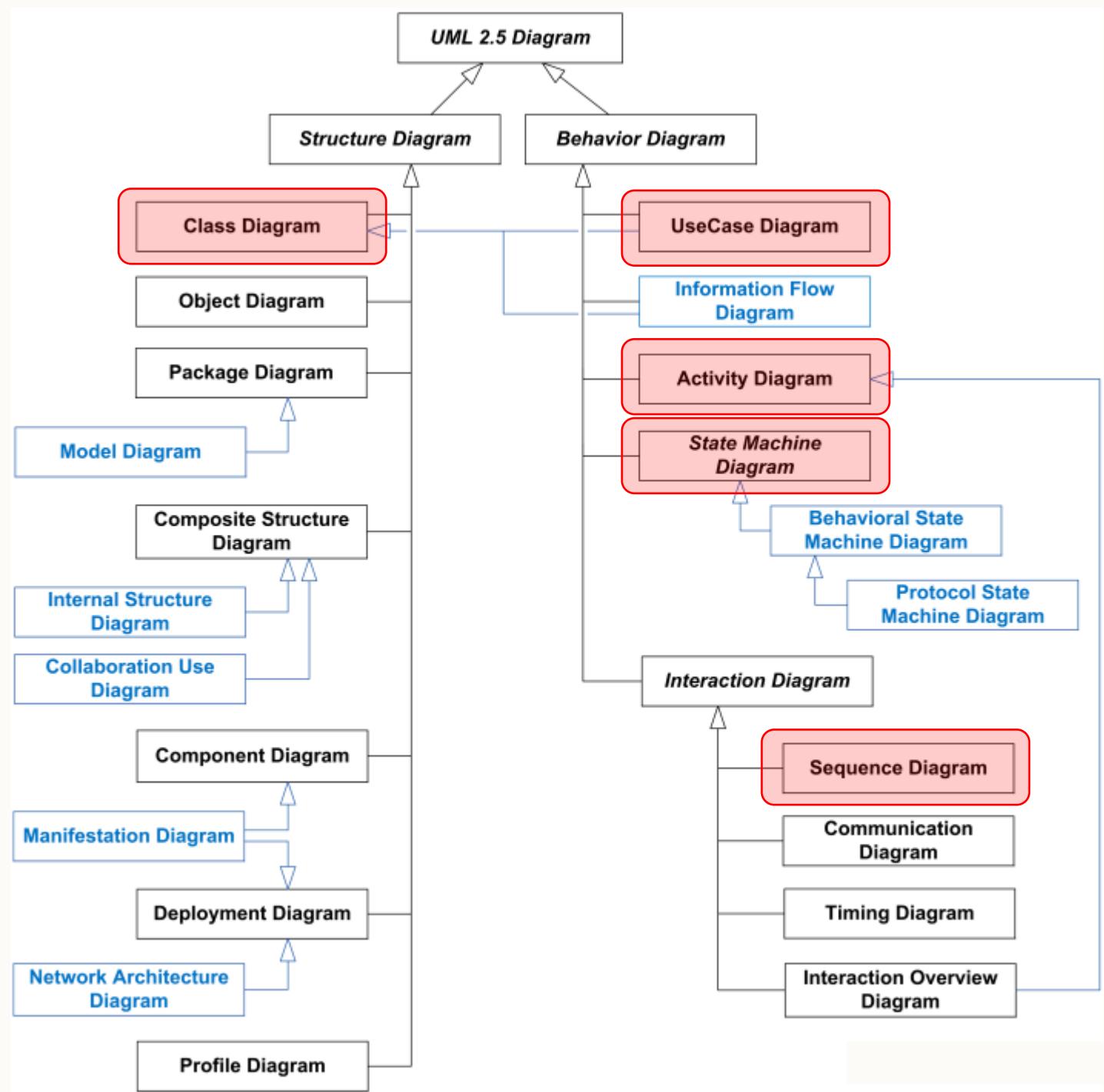
<https://www.omg.org/spec/UML/About-UML/>

- Latest Version: 2.5.1 (Dec 2017)



The screenshot shows the official OMG (Object Management Group) website. The top navigation bar includes links for About Us, Resource Hub, OMG Specifications, Programs, Membership, and Members Area. Below this, a specific page for 'ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1' is displayed. The page title is 'Unified Modeling Language'. A brief description states: 'A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.' Technical details listed include: Title: Unified Modeling Language, Acronym: UML®, Version: 2.5.1, Document Status: formal, Publication Date: December 2017, Categories: Modeling, Software Engineering, and IPR Mode: RF-Limited. A small icon labeled 'Specification' is also visible.







1



2

How the customer explained it

How the programmer wrote it.

How the customer really needed.

How the customer was billed.

3



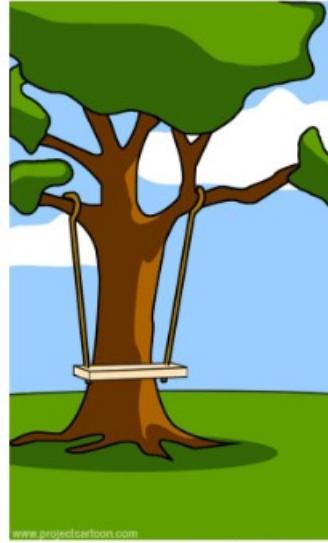
4



Source: <https://medium.com/@thx2001r/the-project-cartoon-root-cause-5e82e404ec8a>



How the customer explained it



How the project leader understood it



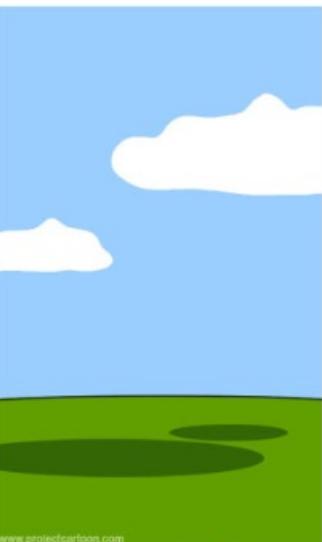
How the analyst designed it



How the programmer wrote it



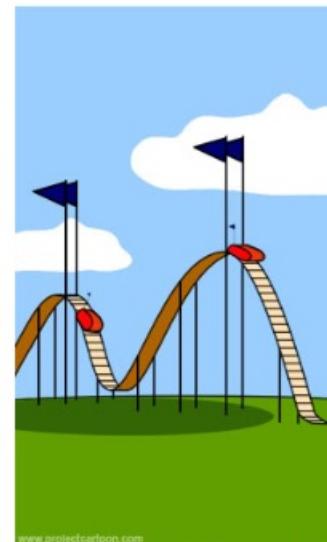
How the business consultant described it



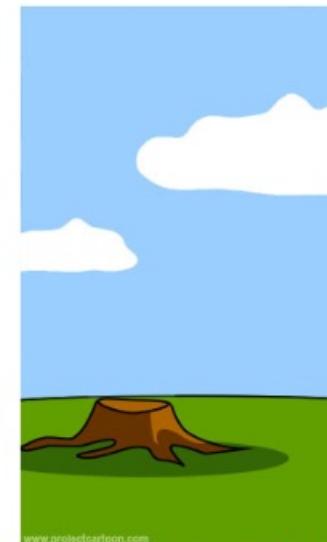
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

Source: <http://www.projectcartoon.com/>



Why UML?

- Advantages of using UML:



COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



Why UML?

- Advantages of using UML:
 - Enhances communication and ensures the right communication
 - Captures the logical software architecture independent of the implementation language
 - Helps to manage the complexity
 - Abstraction
 - Enables reuse of design

COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



Object oriented analysis and design process

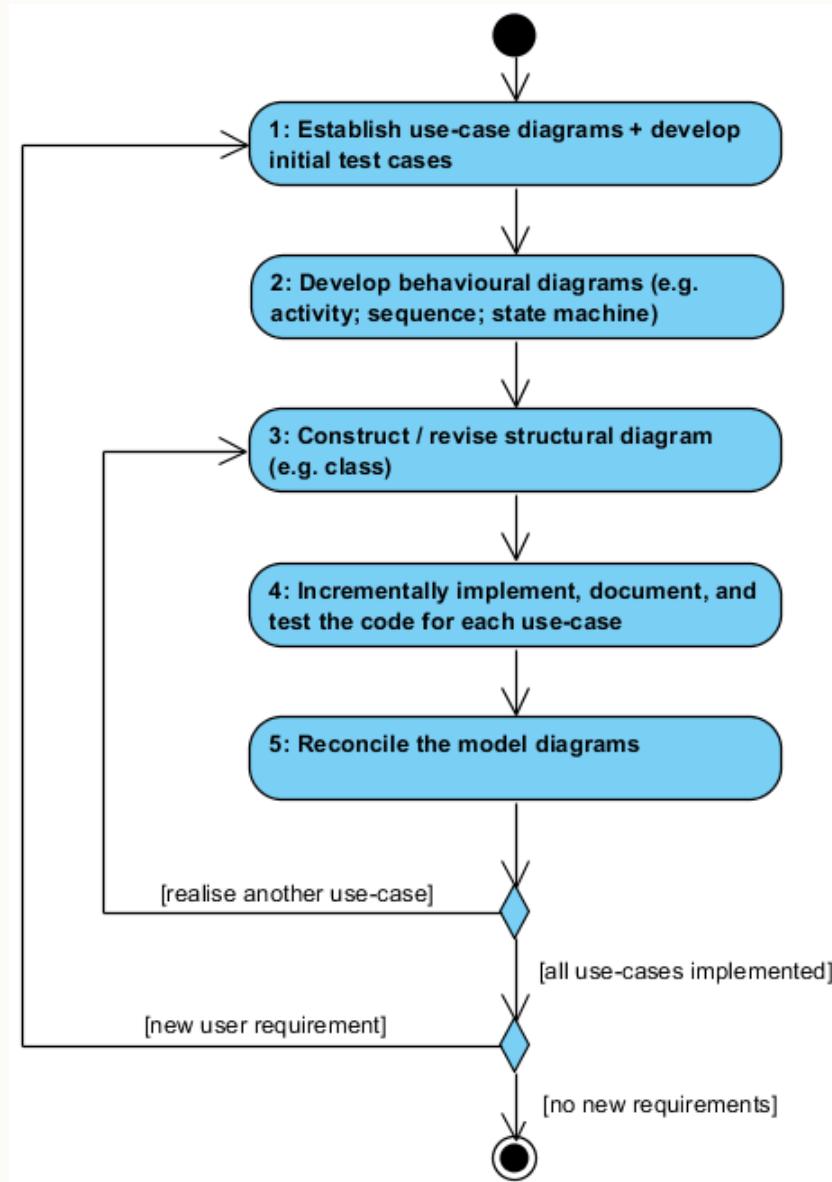
COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



"Use Case Driven" OOA/D Process



[after Barclay and Savage 2004]



Case Study 1

Library Booking System

COMP2013: Developing Maintainable Software
Week 5 – 4:00pm Monday – 23 October 2023



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



Case Study 1: Library Booking System





Case Study 1: Library Booking System

- Library Rules
 - The library contains books and journals; it may have several copies of a given book; some are for short term loan only; the others can be borrowed by any library member for three weeks
 - Normal members can borrow up to 6 books at the same time, staff members up to 12
 - Only staff members can borrow journals
- System Requirement
 - The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above



Case Study 1: Library Booking System

- Our Job: Development of a Library Booking System
- After discussing priorities with the university we decided that the first iteration of the system should consider the following **user stories**:
 - As a university member I want to be able to borrow a copy of a book
 - As a university member I want to be able to return a copy of a book
 - As a university staff member I want to be able to borrow a journal
 - As a university staff member I want to be able to return a journal

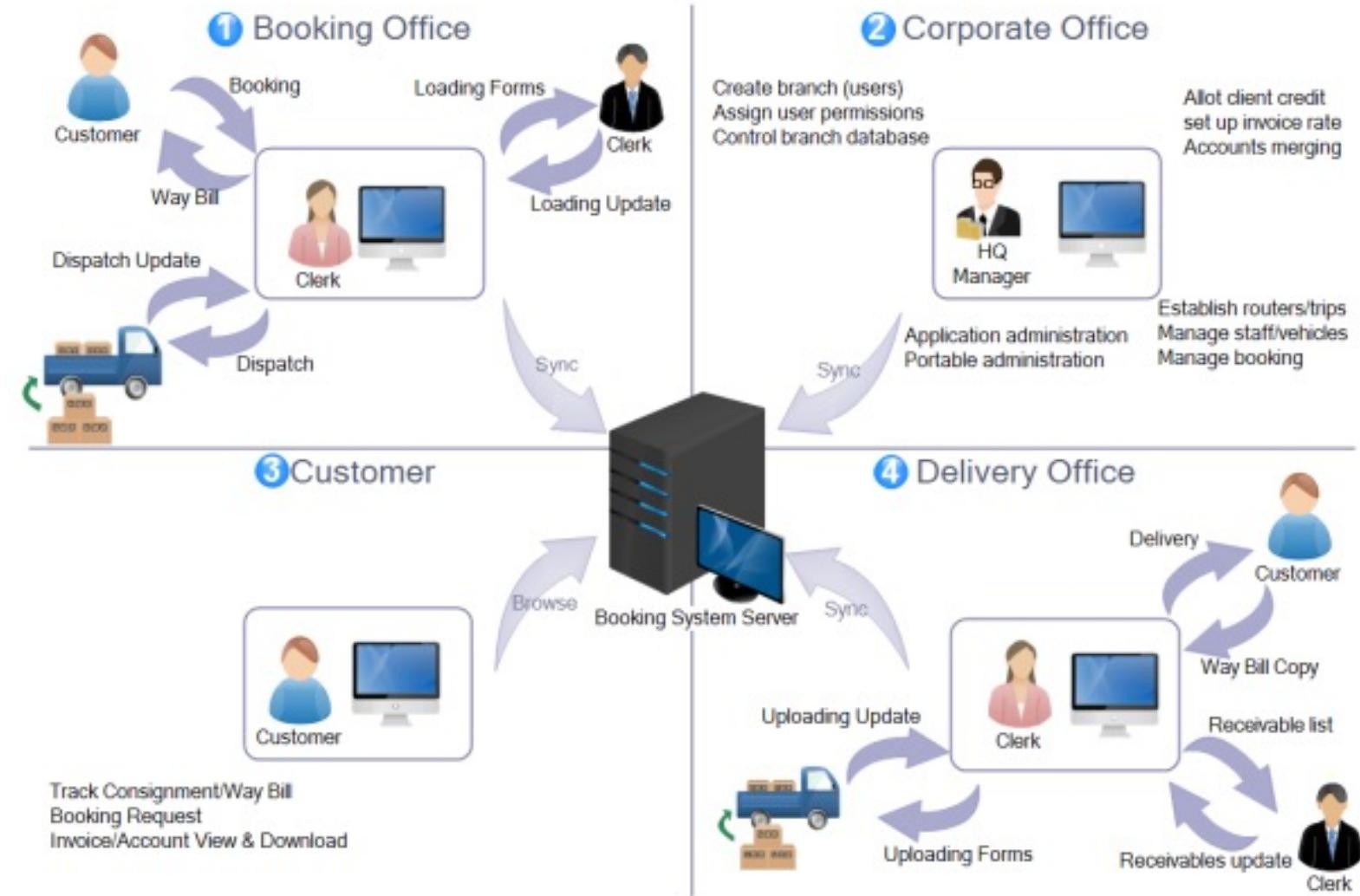


Case Study 2

Fleet Logistics Management



Case Study 2: Fleet Logistics Management





Case Study 2: Fleet Logistics Management

- User stories

- As a client I want to be able to check availability of lorries
- As a client I want to be able to track cargo
- As a manager I want to be able to see the finances
- As an admin I want to be able to search for information
- As an admin I want to be able to organise routes
- As an admin I want to be able to track lorries and cargo





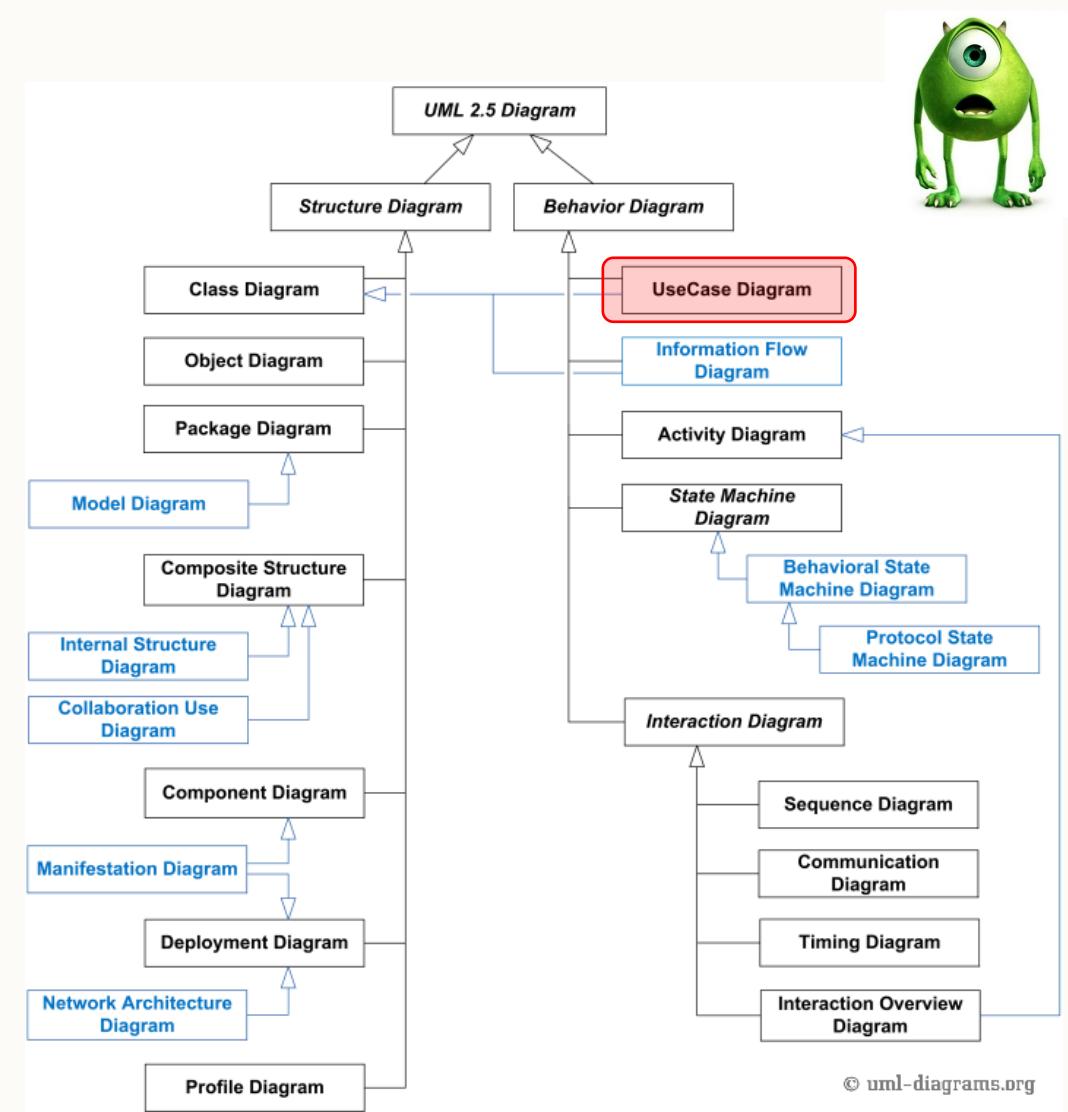
Object oriented analysis

Fleet Logistics Management



Components Covered

- We will be looking at:
 - Use case diagrams
 - Use case specifications
- Are use case diagrams structure or behaviour diagrams?



© uml-diagrams.org



Use Case Diagrams

- What are use case diagrams used for?
 - They describe a set of actions that some system or systems should or can perform in collaboration with one or more external users of the system or systems
 - No attempt to represent an order or a number of executions
- Use case diagram components
 - Actors
 - Use cases
 - Subject (also referred to as "system boundary")
 - Relationships



Use Case Diagrams

- Actors
 - Entities that interface with the system
 - Can be people or other systems
- Use cases
 - Based on user stories
 - Represent what the actor wants your system to do for them
 - In the use case diagram only the use case name is represented
 - In a use case specification each use case is formally described



Use Case Diagrams

- **Subject** (also referred to as "system boundary")
 - Classifier representing a business, software system, physical system or device under analysis, design, or consideration
- **Relationships**
 - Relationship between use case and actor:
 - Association indicates which actor **initiates** which use case
 - Relationship between two use cases:
 - Specifying common functionality and simplifying use case flows
 - Using <<include>> or <<extend>>



Use Case Diagrams

- <<include>>
 - Used when multiple use cases share a piece of same functionality which is placed in a separate use case
 - Arrow points to the more specific use case

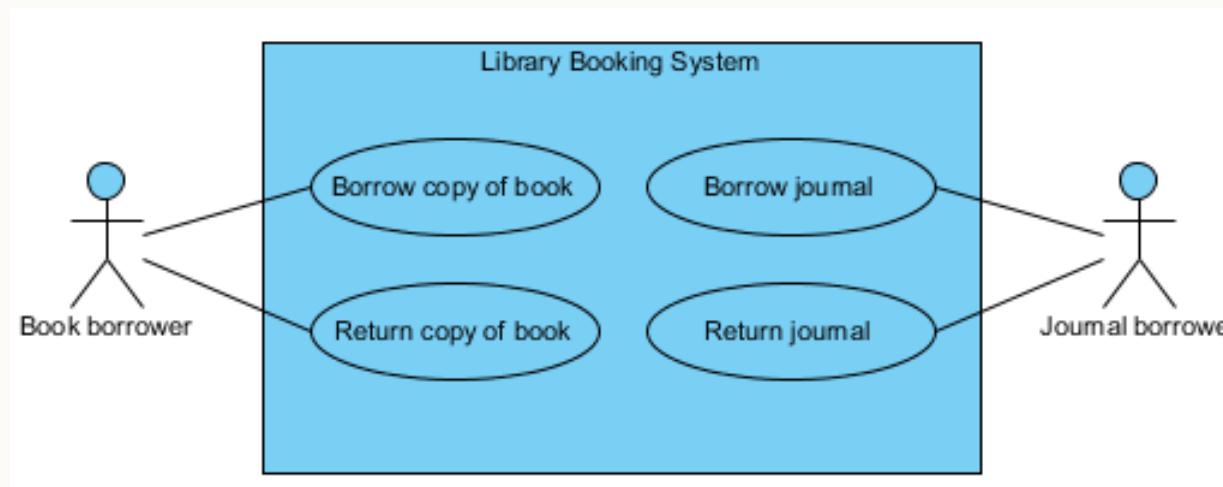
- <<extends>>
 - Used when activities might be performed as part of another activity but are not mandatory for a use case to run successfully
 - Arrow points to the more general use case



Example: Library Booking System

- Reminder

- The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals





Activity: Fleet Logistics Management

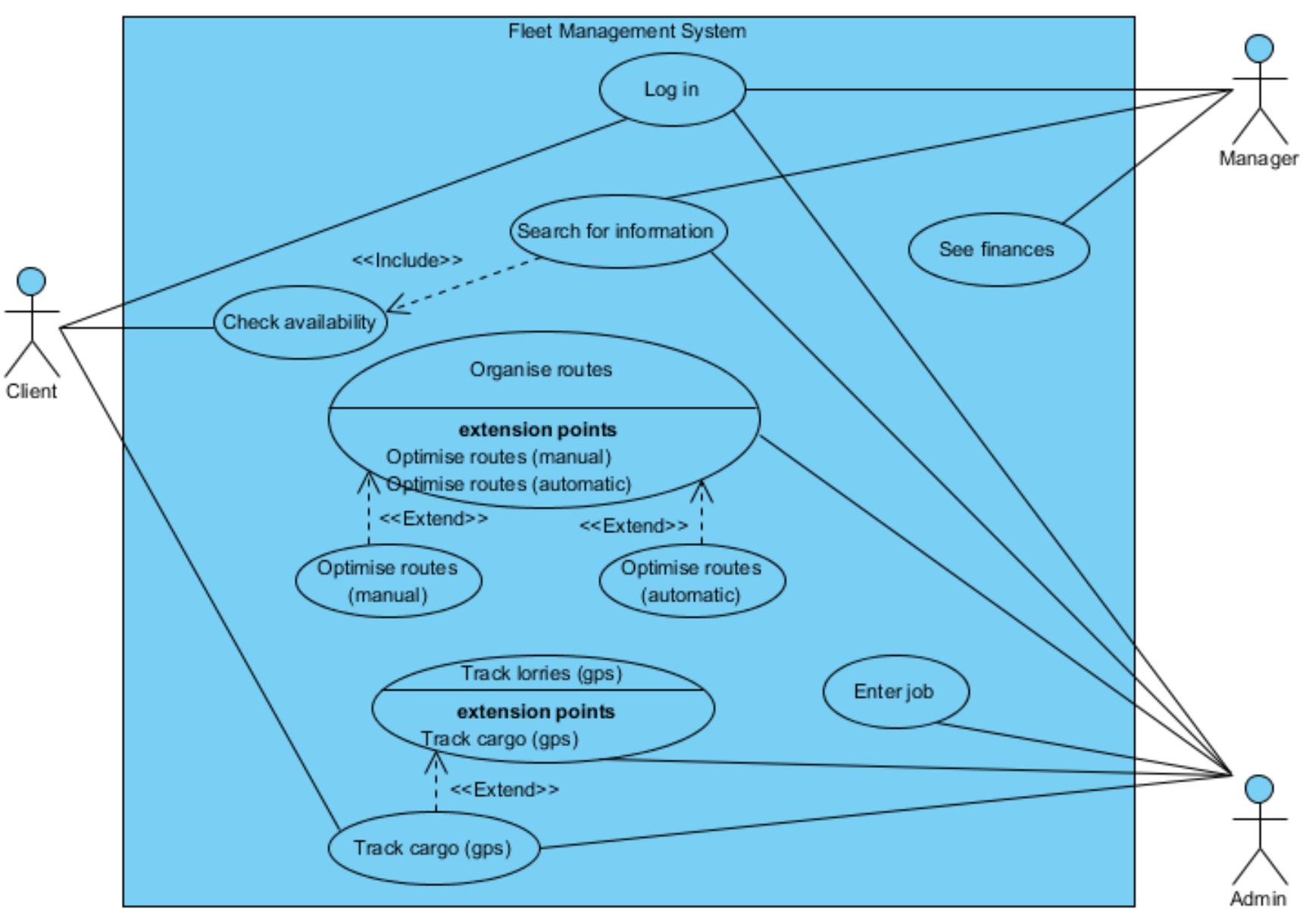
- Use Case Diagram?



- Reminder

- Client wants to check availability of lorries and track cargo.
Manager wants to see the finances. Admin wants to search for information, organise routes and track lorries and cargo



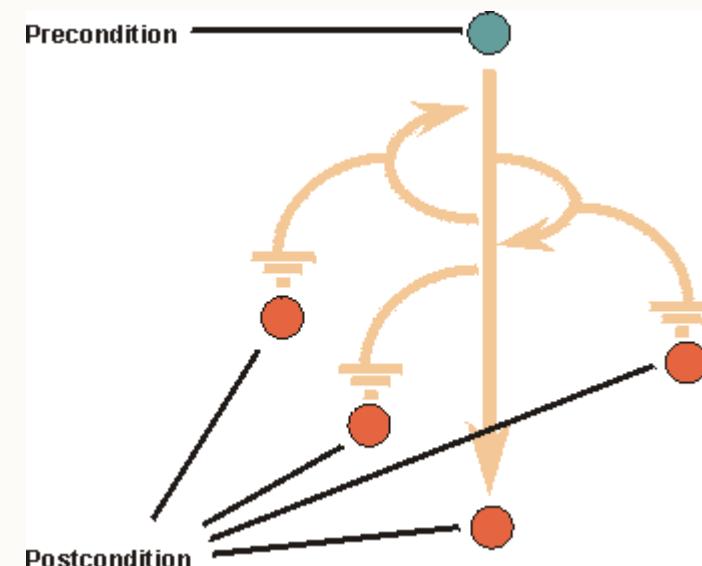




Use Case Specification

- Use case specification elements

- Use case name
 - Use case purpose
 - Pre-condition(s)
 - Base Path
 - Alternative Paths
 - Post-condition(s)





Use Case Specification

- Base and alternative path:
 - Base Path (optimistic flow)
 - "happy day" scenario
 - Alternative Paths (pragmatic flows)
 - Every other possible way the system can be (ob)used
 - Includes perfectly normal alternative use, but also errors and failures



Example: Library Booking System

- Use Case: Borrow copy of books
 - Purpose:
 - The Book Borrower borrows a book from the library using the Library Booking System
 - Pre-condition(s):
 - The book must exist
 - The book must be available



Example: Library Booking System

- Use Case: Borrow copy of books
 - Base Path (optimistic flow)
 1. LBS requests membership card
 2. BB provides membership card
 3. BB is logged in by LBS
 4. LBS checks permissions / debts
 5. LBS asks for presenting a book
 6. BB presents a book
 7. LBS scans RFID tag inside book
 8. LBS updates records accordingly
 9. LBS disables anti-theft device
 10. BB is logged out by LBS
 11. LBS confirms that process has been completed successfully

BB = Book Borrower
LBS = Library Booking
System



Example: Library Booking System

- Use Case: Borrow copy of books
 - Alternative Paths (pragmatic flows)
 1. BB's card has expired: Step 3a: LBS must provide a message that card has expired; LBS must exit the use case
 2. ...
 - Post-condition(s) for base path:
 - Base Path
 - BB has successfully borrowed the book; the LBS is up to date
 - Alternative Path 1
 - BB was NOT able to borrow the book; the LBS is up to date
 - Alternative Path 2
 - ...

BB = Book Borrower

LBS = Library Booking
System



Activity: Fleet Logistics Management

- Use Case Specification for use case "Search for Information"?



- Reminder

- Client wants to check availability of lorries and track cargo.
Manager wants to see the finances. Admin wants to **search for information**, organise routes and track lorries and cargo



- Use case specification elements
 - Use case name
 - Use case purpose
 - Pre-condition(s)
 - Base Path
 - Alternative Paths
 - Post-condition(s)



Activity: Fleet Logistics Management

- Use Case: Search for Information
- Purpose:
 - Admin can search the DB for any kind of information related to lorries and jobs
- Pre-condition(s):
 - Admin is logged in

DB = Database



Activity: Fleet Logistics Management

- Use Case: Search for Information
 - Base Path (optimistic flow)
 1. Admin opens search window
 2. Admin defines query using query editor
 3. Admin sends query to DB
 4. DB deals with query: finding results
 5. DB deals with query: organising them by relevance
 6. DB sends results back
 7. DB requests confirmation that result is sufficient
 8. Admin confirms that result is sufficient
 9. DB closes search window



Activity: Fleet Logistics Management

- Use Case: Search for Information
 - Alternative Paths (pragmatic flows)
 1. Admin has not received the required information: Step 7a Admin denies that result is sufficient; Admin must go back to Step 2.
 2. DB is not accessible: Step 3a: DB returns warning message that DB is not accessible; use case needs to be left
 - Post-condition(s):
 - Base Path and Alternative Path 1
 - The admin has retrieved the required information
 - Alternative path 2
 - The admin has NOT retrieved the required information



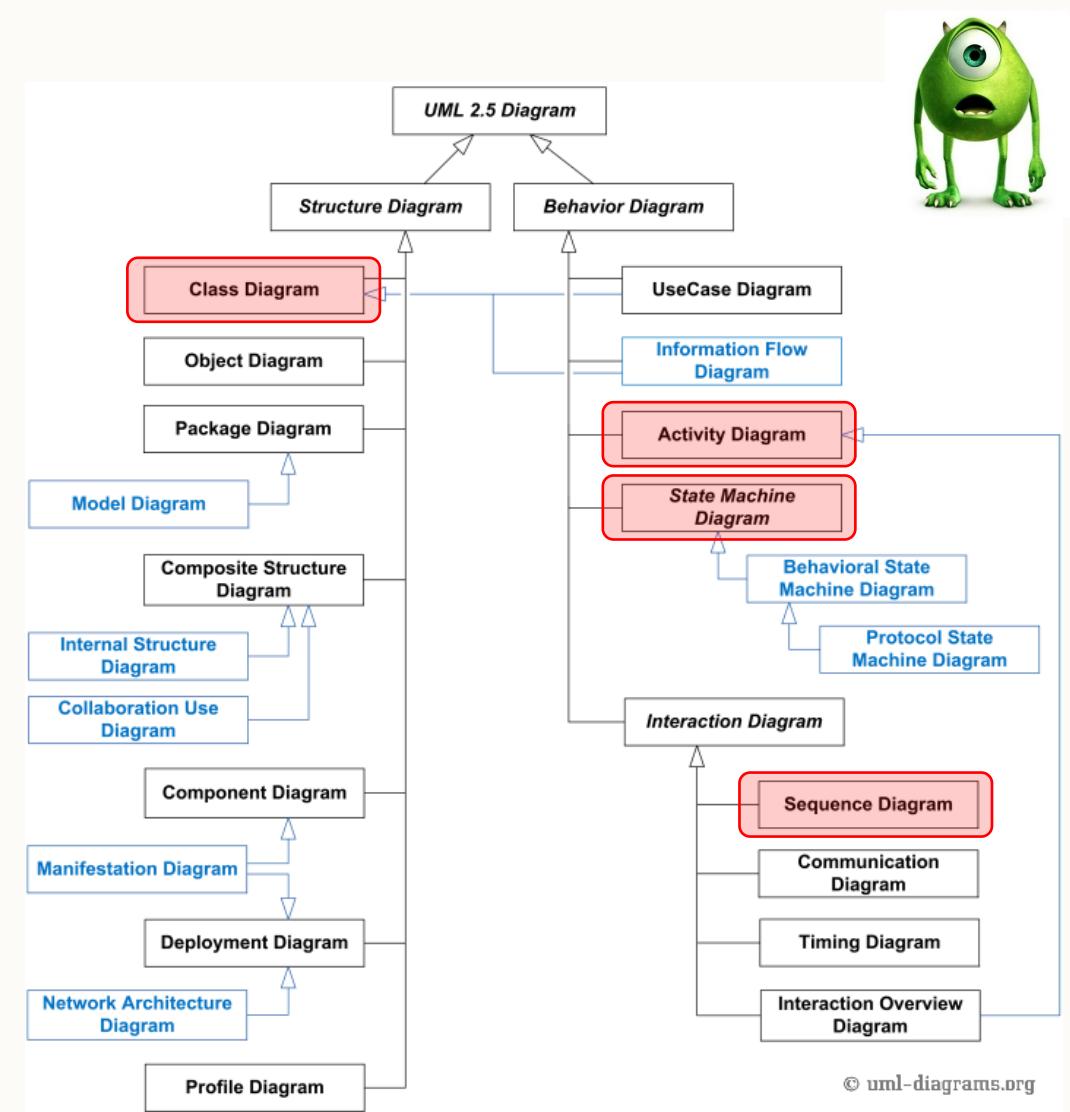
Object oriented design

Fleet Logistics Management



Components Covered

- We will be looking at:
 - Activity diagrams
 - Sequence diagrams
 - State machine diagrams
 - Class diagrams
- Which of these are structure / behaviour diagrams?



© uml-diagrams.org



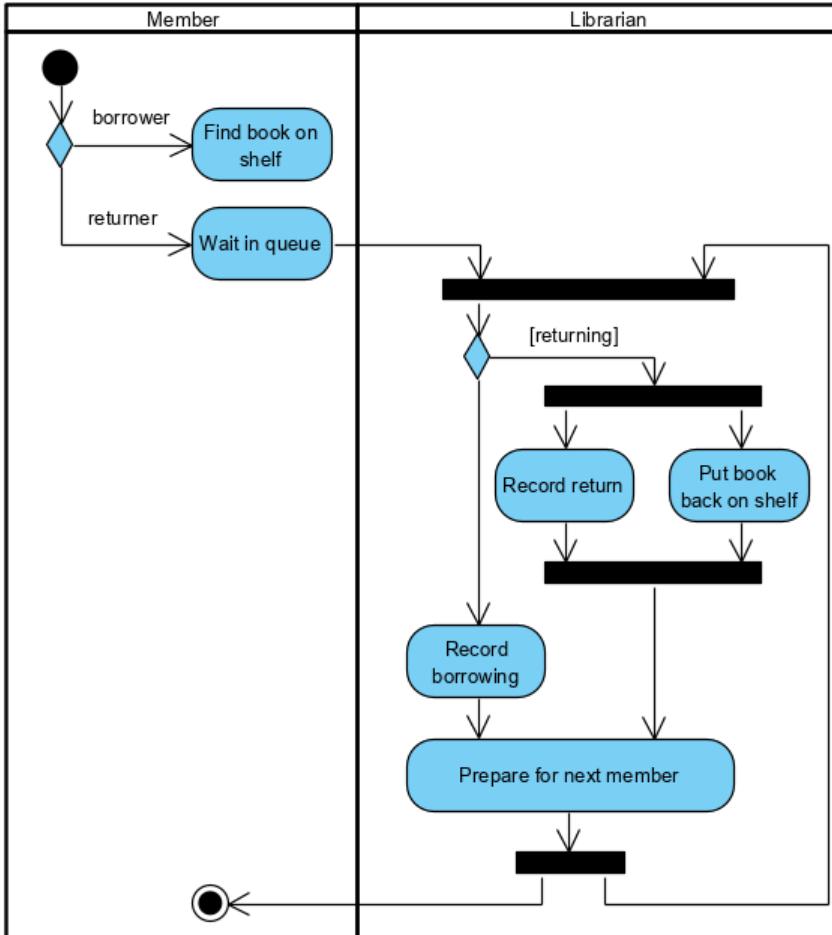


Activity Diagrams

- What are activity diagrams used for?
 - Graphical representations of workflows of stepwise activities and actions related to an individual use case or across many use cases
 - Support representation of parallel behaviour
- Activity diagram components
 - Activity: A state that is left once the activity is finished
 - Activity edge: Transition that fires when previous activity completes
 - Synchronisation bar: Describes the co-ordination of activities
 - Decision diamond: Used to show decisions
 - Start and stop marker: Used to define entry and exit points
 - Swim lane: A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread



Example: Library Booking System





Activity: Fleet Logistics Management

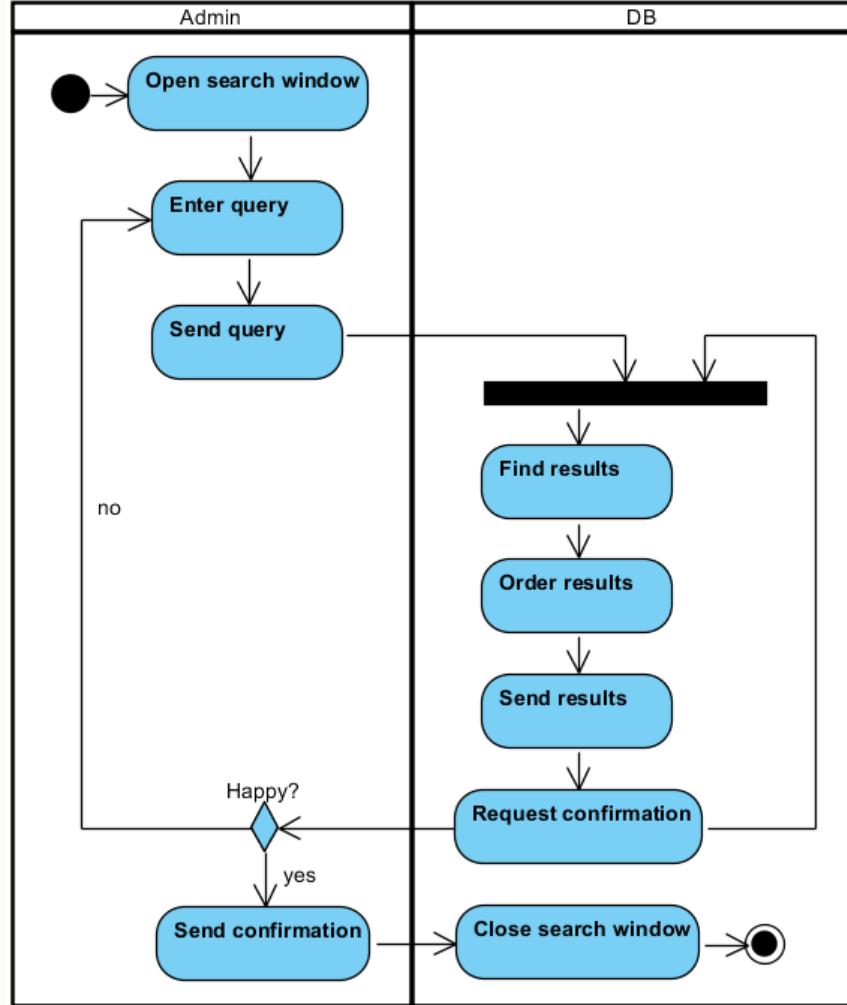
- Activity Diagram for use case "Search for Information"?



- Reminder: Base Path (optimistic flow) for this use case
 1. Admin opens search window
 2. Admin defines query using query editor
 3. Admin sends query to DB
 4. DB deals with query: finding results
 5. DB deals with query: organising them by relevance
 6. DB sends results back
 7. DB requests confirmation that result is sufficient
 8. Admin confirms that result is sufficient
 9. DB closes search window



Activity: Fleet Logistics Management





Sequence Diagrams

- What are sequence diagram used for?
 - Sequence diagrams are a temporal representation of objects and their interactions
- Sequence diagram components
 - Participant: Object or actors that act in the sequence diagram
 - Vertical line (called lifeline): Represent time as seen by the object
 - Narrow rectangle covering an object's life line shows a live activation of the object
 - Arrow from sender's lifeline to receiver's lifeline:
 - General: Message, denoting an event or the invocation of an operation
 - Object creation: Arrow with 'new' written above it
 - Object deletion: An X at bottom of lifeline (in some OOP languages this is handled automatically)
 - Sequence fragment: Let you show loops, branches, and other alternatives

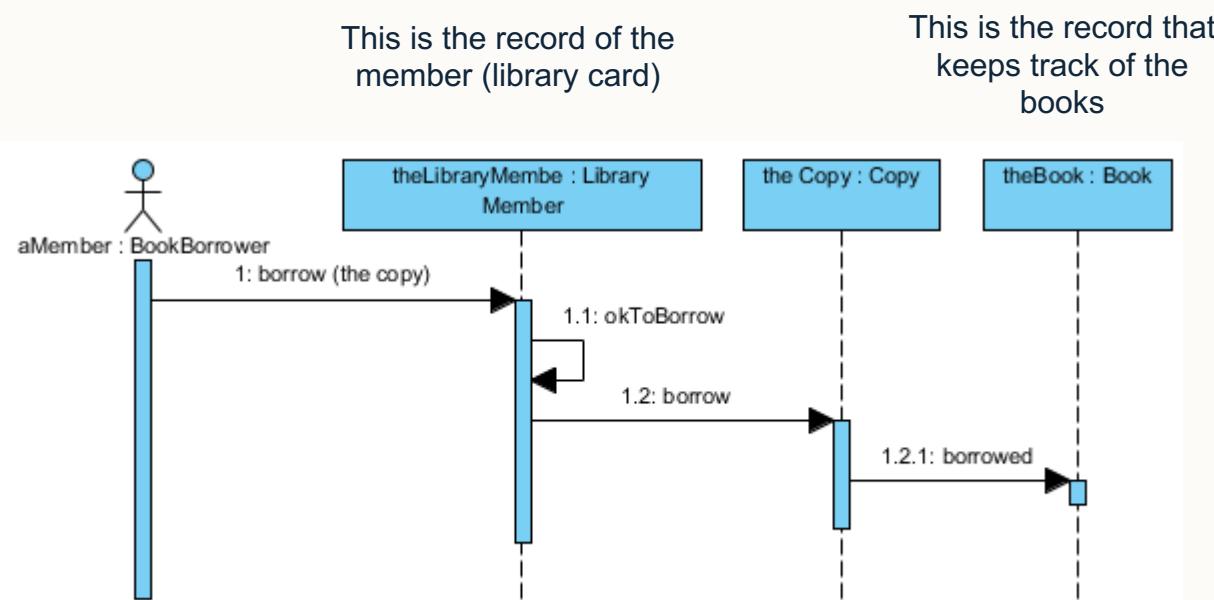


Example: Library Booking System

-

Reminder

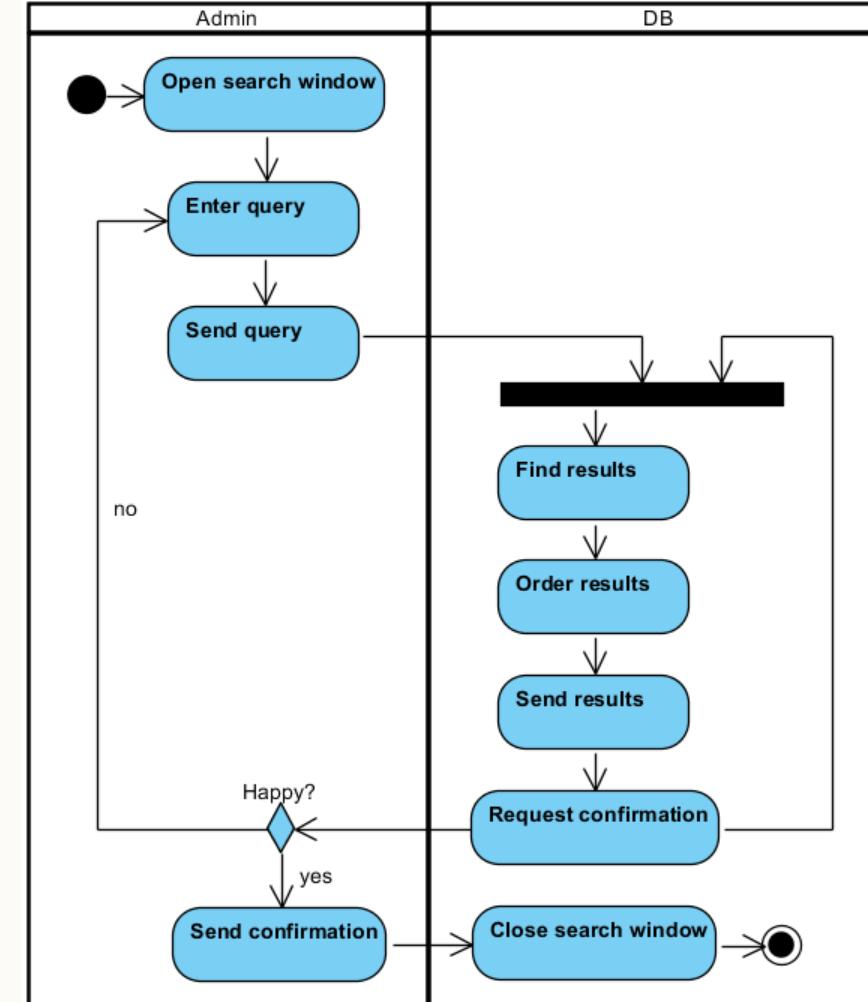
- The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals





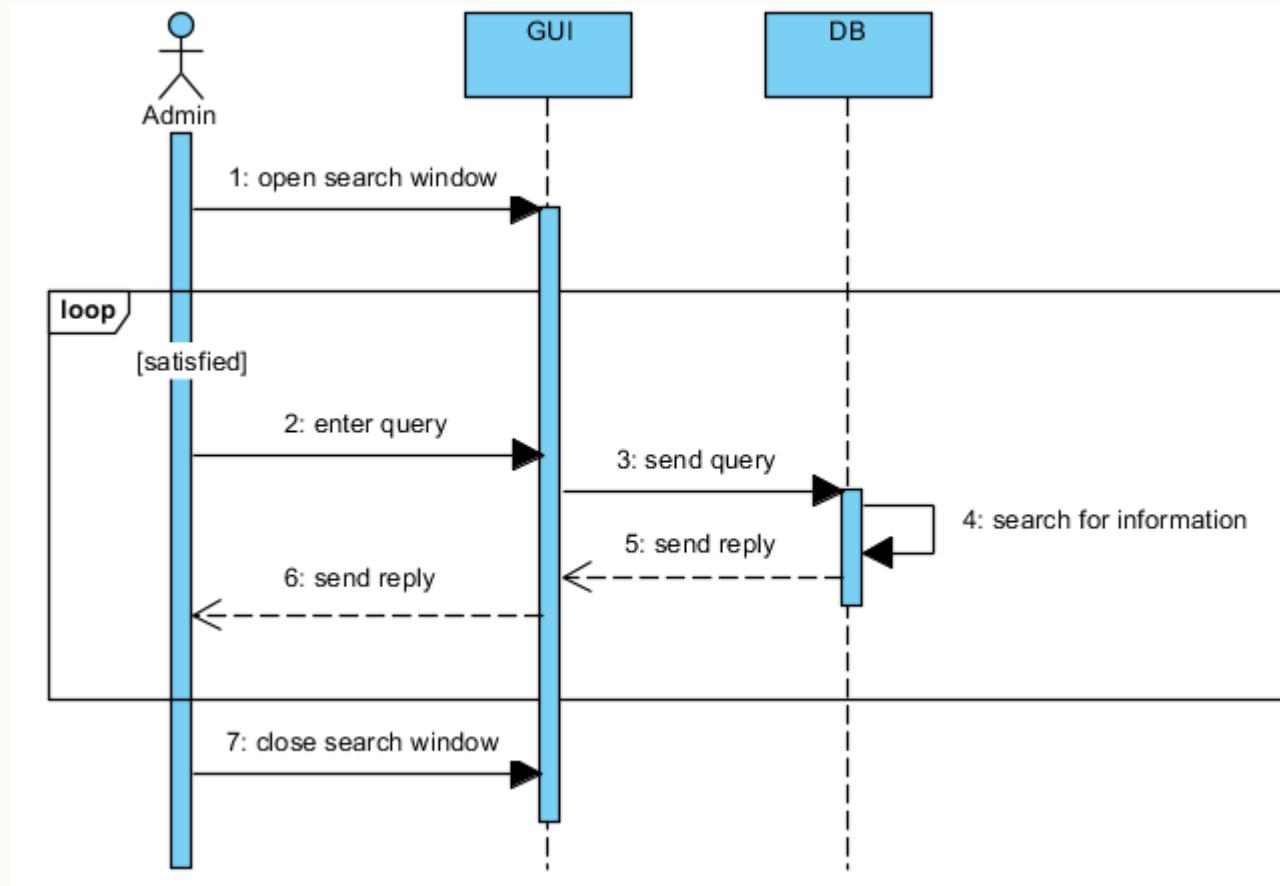
Activity: Fleet Logistics Management

- Sequence Diagram for use case "Search for Information"?
- Reminder: Activity Diagram





Activity: Fleet Logistics Management





State Machine Diagrams

- What are state machine diagrams used for?
 - Show the possible **states of a single object**, the events or messages that cause a transition from one state to another, and the action that result from that state change
 - Only reactive objects require a state machine diagram!
- State machine diagram components
 - State: A condition during the life of an object when it satisfies some condition, performs some action, or waits for an event.
Special states:
 - Start state: Each state diagram must have one and only one start state
 - Stop state: An object can have multiple stop states



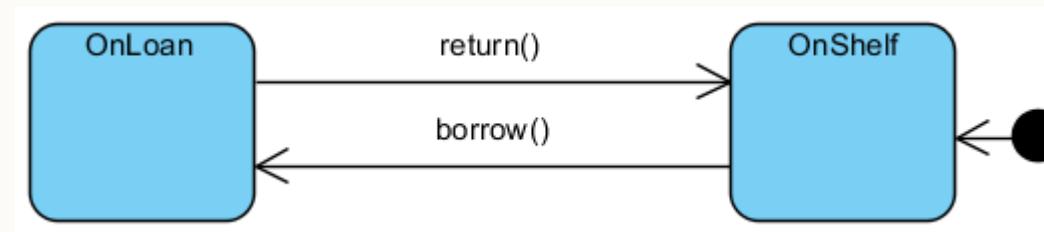
State Machine Diagrams

- State machine diagram components (continued)
 - Transition: Denotes the transition between states or to the same state (self-transition)
 - A transition may have a trigger, a guard and an effect (Trigger[Guard]/Effect):
 - Trigger: The cause of the transition (signal; event; change in some condition; passage of time)
 - Guard: Condition which must be true in order for the trigger to cause the transition
 - Effect: Action invoked directly on the object that owns the state machine as a result of the transition

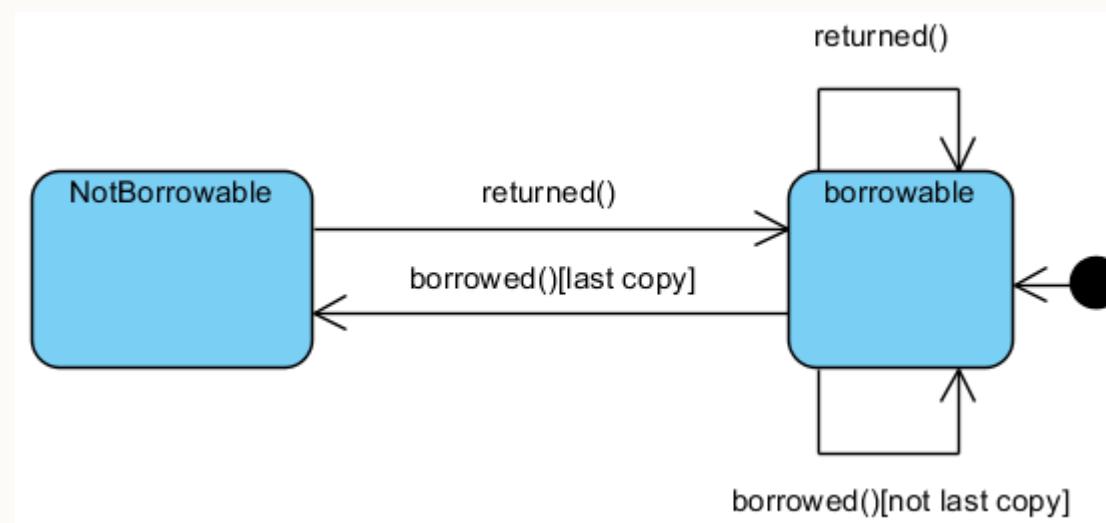


Example: Library Booking System

- State machine diagram for the "Copy" class



- State machine diagram for the "Book" class





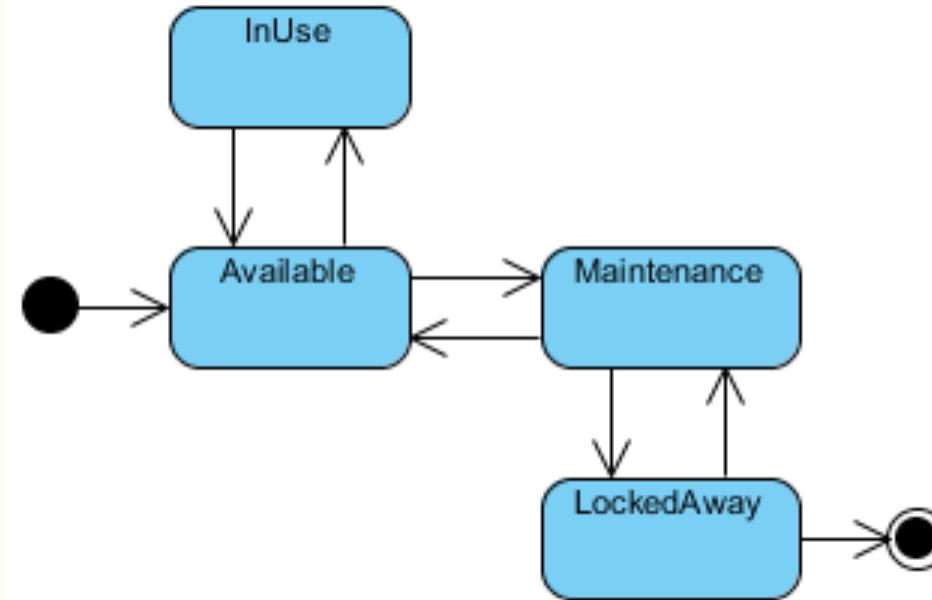
Activity: Fleet Logistics Management

- State Machine Diagram for "Lorry" Class?





Activity: Fleet Logistics Management





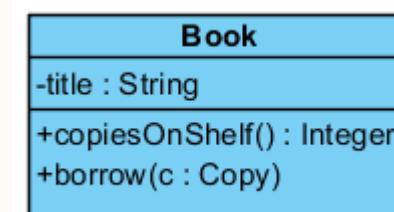
Class Diagrams

- What is a class diagram used for?
 - A class diagrams shows the existence of classes and their structures and relationships in the logical view of a system
- Class diagram's main components:
 - Classes
 - Class relationships
 - Associations; dependencies; aggregations; compositions; realisations; generalisations
 - Multiplicity indicators



Class Diagrams

- Class representation
 - In UML classes are depicted as rectangles with three compartments
 - Class name
 - Attributes: Describe the data contained in an object of the class
 - Operations: Define the ways in which objects interact
 - Additional symbols
 - + public
 - # protected
 - private
 - / derived
 - \$ static



This is the record that
keeps track of the books



Class Diagrams

Associations between classes

- Classes are associated if an instance of class A (the source class) has to know about an instance of class B (the target class) or vice versa

Multiplicity indicators

- Number of links between each instance of the source class and instances of the target class
 - 1 = exactly 1
 - * = unlimited number (zero or more)
 - 0..* = zero or more
 - 1..* = one or more
 - 0..1 = zero or 1
 - 3..7 = specified range (from 3 to 7)



Class Diagrams

- Relationship: Association
 - Reference based relationship between two classes
 - Class A holds a class level reference to class B
 - Represented by a line between A and B with an arrow indicating the navigation direction
 - If no arrow or arrow on the both sides, association has bidirectional navigation

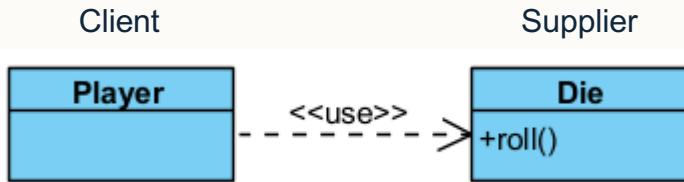


```
1 class Asset {  
2     ...  
3 }  
4  
5 class Player {  
6     private Asset asset  
7     ...  
8     public Player(Asset purchasedAsset) {  
9         this.asset = purchasedAsset;  
10    ...  
11 }  
12 }
```



Class Diagrams

- Relationship: Dependency
 - Created when you receive a reference to a class as part of a particular method
 - Dependency indicates that you may invoke one of the APIs of the received class reference and any modification to that class may break your class as well
 - Multiplicity does not make sense on a Dependency



Supplier-client relationship, where the supplier provides something to the client, and thus the client is in some sense incomplete while semantically or structurally dependent on the supplier element(s). Modification of the supplier may impact the client elements.

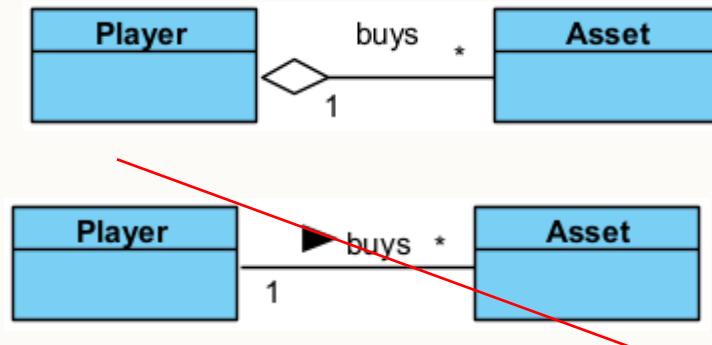
<<use>> The source (Player) requires the target (Die) for the implementation

```
1 class Die {  
2     ...  
3     public void roll() {...}  
4 }  
5  
6 class Player {  
7     ...  
8     public void takeTurn(Die die) {  
9         die.roll();  
10    ...  
11 }  
12 }
```



Class Diagrams

- Relationship: Aggregation ("is part of" relationship)
 - Often seen as redundant relationship as technically the same as an association; but semantically there is a difference
 - It is used when an object logically or physically contains another; the container is called "aggregate"; the components of the aggregate can be shared with others

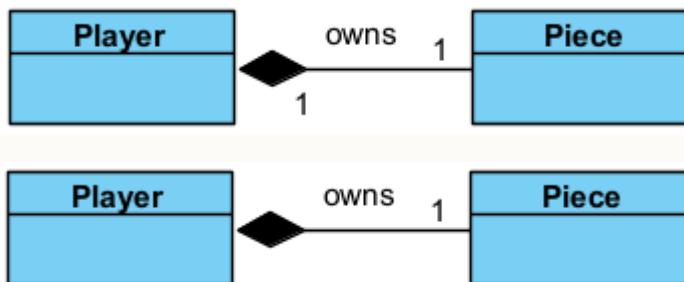


```
1 class Asset {
2     ...
3 }
4
5 class Player {
6     private List assets
7     ...
8     public void addAsset(Asset purchasedAsset) {
9         ...
10        assets.add(purchasedAsset);
11    ...
12 }
13 }
```



Class Diagrams

- Relationship: Composition
 - Relates to instance creational responsibility
 - When class B is composed by class A, class A instance owns the creation or controls lifetime of instance of class B
 - Composition binds lifetime of a specific instance for a given class, while class itself may be accessible by other parts of the system.
 - Multiplicity at the composition end is always 1 as parts have no meaning outside the whole



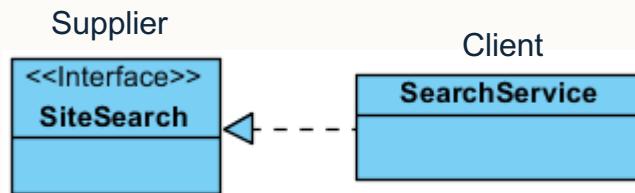
```
1 class Piece {
2 ...
3 }
4
5 class Player {
6     private Piece piece=new Piece();
7 ...
8 }
```



Class Diagrams

- Relationship: Realisation

- A "Realisation" is a specialised abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other representing an implementation (the client) of the specification

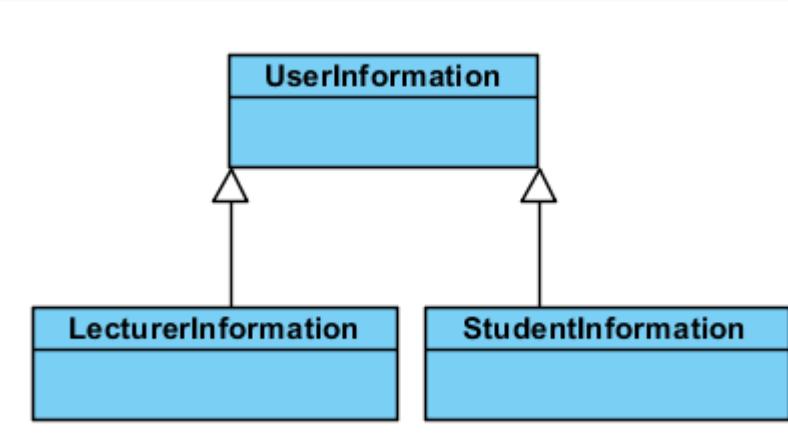


```
1 class SearchService implements SiteSearch {  
2     ...  
3 }
```



Class Diagrams

- Relationship: Generalisation ("is a" relationship)
 - A directed relationship between a more general classifier (superclass) and a more specific classifier (subclass)

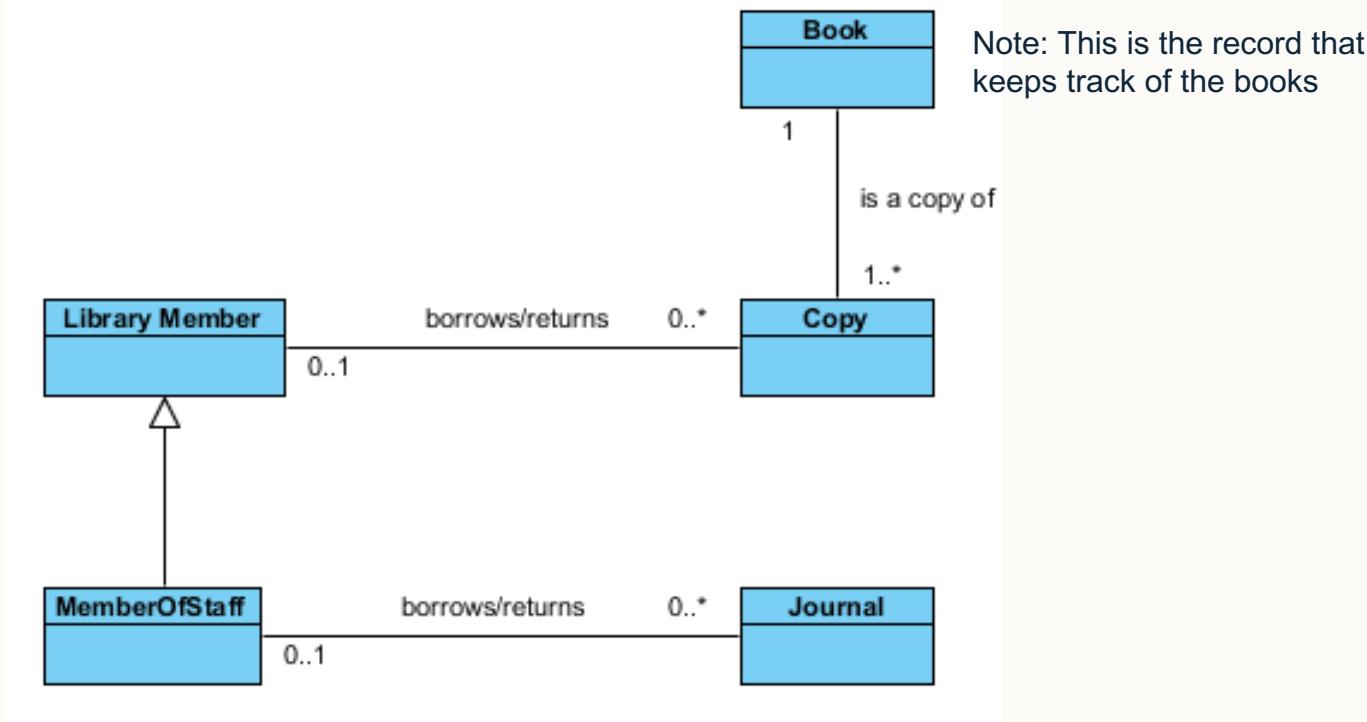


```
1  class LecturerInformation extends UserInformation {  
2      ...  
3  }  
4  
5  class StudentInformation extends UserInformation {  
6      ...  
7  }
```



Example: Library Booking System

- Reminder
 - The library contains books and journals; it may have several copies of a given book; only staff members can borrow journals





Activity: Fleet Logistics Management

- Class Diagram?

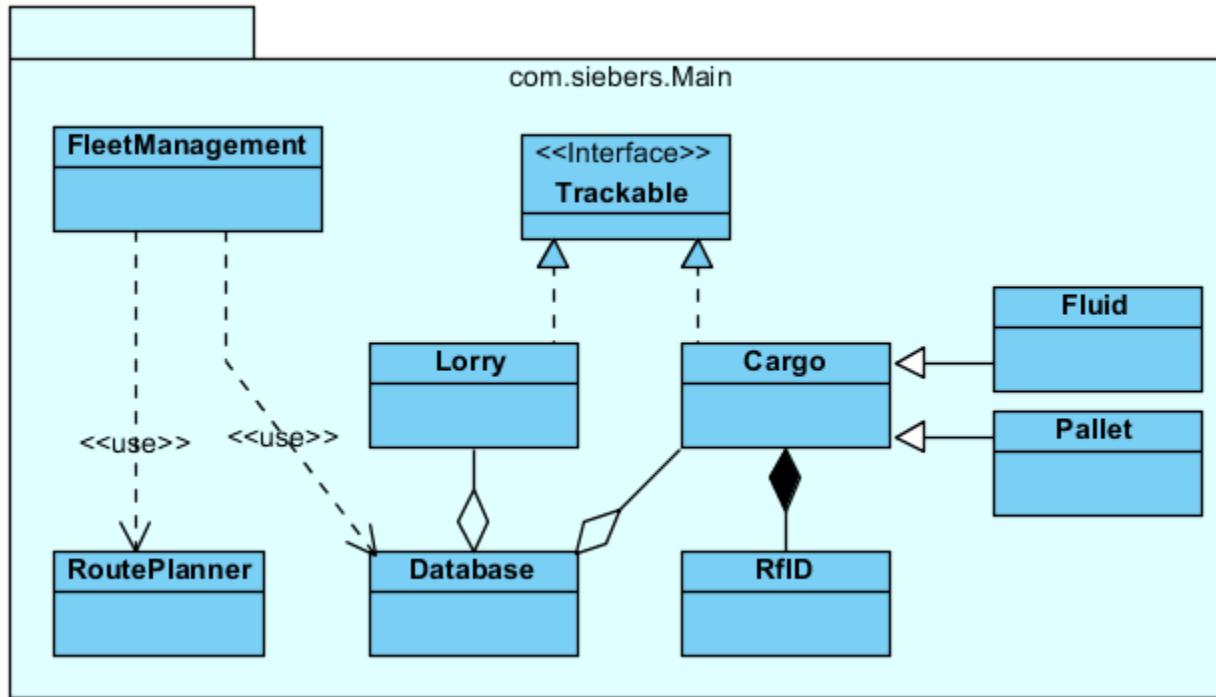


- Reminder

- Client wants to check availability of lorries and track cargo.
Manager wants to see the finances. Admin wants to search for information, organise routes and track lorries and cargo



Activity: Fleet Logistics Management





Acknowledgement

- Slides based on material from
 - Peer-Olaf Siebers's lecture slides 2022/2023
- Barclay and Savage (2004) Object-Oriented Design with UML and Java
- Some of the relationship descriptions taken from
<https://nirajrules.wordpress.com/2011/07/15/association-vs-dependency-vs-aggregation-vs-composition/>
- A good read on interface and inheritance
<https://medium.com/@marcomvidal/oop-when-using-interfaces-751c74767b8c>



The University of
Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Lecture 04B

Introduction to Maven and Gradle



A nighttime photograph of a modern university building complex. The buildings have a warm, glowing interior with many windows lit up. The architecture is a mix of angular and curved structures, some with wooden cladding. The building is reflected in a dark body of water in the foreground. The sky is a deep blue of twilight.

Horia A. Maior and Marjahan Begum

</>



Lecture 04B

Introduction to Maven and Gradle

Horia A. Maior and Marjahan Begum

Topics for this Week

</>

- Lecture 04A:
 - OO Analysis and Design (OOA/D) with UML
- Lecture 04B:
 - Early Module Feedback (5 min)
 - Week 4 Labsheet
 - Build Tools (Maven and Gradle)
 - Coursework Preview
- Lab 04 (Tomorrow):
 - Virtual UML Speed Refactor/Extend Challenge for Group Project groups

</>

Module Assessment Sheet 2023-24



Module Information

Timetable of Activities

Teaching Week	W/C	Topic	Lead	Format		
				Lectures Mondays, 4pm-6pm	Labs Fridays 3-5pm	
1	02/10/2023	Introduction to DMS	Horia/Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	
2	09/10/2023	More Advanced Java Topic	Horia	BS South B52 + Recording	Lab A07,A32,Atrium	
3	16/10/2023	Maintainable GUI Development (1/2)	Horia	BS South B52 + Recording	Lab A07,A32,Atrium	
4	23/10/2023	Design Principles and Patterns	Horia/Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	
5	30/10/2023	Maintainable GUI Development (2/2)	Horia/Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	
6	06/11/2023	Coding and Repository Tools for DMS	Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	
7	13/11/2023	UML for the Maintainer	Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	
8	20/11/2023	Refactoring Skills	Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	
9	27/11/2023	Open Source	Horia	BS South B52 + Recording	Lab A07,A32,Atrium	
10	04/12/2023	Guest Lecture	Horia/Marjahan	BS South B52	Lab A07,A32,Atrium	
11	11/12/2023	Revision and Exam Prep	Marjahan/Horia	BS South B52 + Recording	Lab A07,A32,Atrium	

Last updated on 29/09/2023.

Module Assessment Sheet

[Module Assessment Sheet] is now available!

Early Module Feedback

[COMP2013 Early Module Feedback 2023-24](#)

Lecture Recordings

Links to recordings will be available, just beside/below each lecture/lab title.

Microsoft Teams Access

If you do not have access to COMP2013 - Developing Maintainable Software (COMP2013 UNUK) in Microsoft Teams. Please use the [link](#) to participate

Module Convenor(s)	Horia Maior, Marjahan Begum
Module Code	COMP2013
Module Credits	20

Term-time Assessment (TTA)

Assessment Name	TTA1 --- Coursework
Assessment Type	Coursework
Description and Deliverable(s)	<p>Description: Maintaining and extending existing software</p> <p>Deliverables: Git activity; refactored and extended code base on GitLab; documentation (readme.md, Javadoc, class diagram reflecting changes); zip file of your project; video showing the game in action and explaining your maintenance work</p>

B: Frequency, dates & workload	<input checked="" type="checkbox"/> Individual <input type="checkbox"/> Weekly <input type="checkbox"/> Fortnightly <input type="checkbox"/> Custom
	Release date: 31/10/2023 Submission date: 12/12/2023 Workload (h): 75
	<input checked="" type="checkbox"/> UoN Default late policy. <input type="checkbox"/> Custom: Enter details of custom late policy (if applicable)

B: Late Policy	<input checked="" type="checkbox"/> Within 15 working days of submission <input type="checkbox"/> The expected date for feedback is Please enter date here.
	<p>Feedback mechanism: We aim to provide written individual feedback in Moodle within 15 working days of deadline, but might need more time due to the class size.</p>

B: Feedback Mechanism & Date	<p>We plan to split marks as follows: 10% for git use (e.g. push, branch, merge, providing .gitignore) 30% for refactoring 30% for additions 10% for documentation (readme.md, Javadoc, class diagram) 20% for the demonstration video, git, documentation, showing the software running, and explaining your refactoring activities and your additions</p>
	<p>Note that ECs are granted based on personal circumstances and are not guaranteed.</p>

ECs	<input type="checkbox"/> Extensions are not possible due to practical limitations <input type="checkbox"/> Assessment component can be disregarded in exceptional circumstances: Please specify details <input checked="" type="checkbox"/> Up to a maximum of 1 week per 10 credits (actual extension given will depend on specific circumstances) <input type="checkbox"/> Cut-off date for extensions applies: Please specify date <input type="checkbox"/> Other: Please provide details
-----	--

Exam (E)

Examination Type	E1 --- In person ExamSys
Weight (%)	25
Duration (h)	1.5
Other Information	Other information

</>

Early Module Feedback 2023-24 is now open

Timetable of Activities

Teaching Week	W/C	Topic	Lead	Format		
				Lectures Mondays, 4pm-6pm	Labs Fridays 3-5pm	Thursdays 10am-12pm
1	02/10/2023	Introduction to DMS	Horia/Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
2	09/10/2023	More Advanced Java Topic	Horia	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
3	16/10/2023	Maintainable GUI Development (1/2)	Horia	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
4	23/10/2023	Design Principles and Patterns	Horia/Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
5	30/10/2023	Maintainable GUI Development (2/2)	Horia/Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
6	06/11/2023	Coding and Repository Tools for DMS	Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
7	13/11/2023	UML for the Maintainer	Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
8	20/11/2023	Refactoring Skills	Marjahan	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
9	27/11/2023	Open Source	Horia	BS South B52 + Recording	Lab A07,A32,Atrium	BS South
10	04/12/2023	Guest Lecture	Horia/Marjahan	BS South B52	Lab A07,A32,Atrium	BS South
11	11/12/2023	Revision and Exam Prep	Marjahan/Horia	BS South B52 + Recording	Lab A07,A32,Atrium	BS South

Last updated on 29/09/2023.

Module Assessment Sheet

[[Module Assessment Sheet](#)] is now available!

Early Module Feedback



[COMP2013 Early Module Feedback 2023-24](#)

Lecture Recordings

Links to recordings will be available, just beside/below each lecture/lab title.

We want to hear from you!

Survey is anonymous.

Microsoft Teams Access

If you do not have access to COMP2013 - Developing Maintainable Software (COMP2013 UNUK) in Microsoft Teams. Please use the [link](#) to participate.

</>

Week 4 Labsheet

Week 4 labsheet

</>

- Practice your UML Design skills
- Work in your already existing SE Groups from COMP2002
- Exercise useful for your coursework
- Exercise useful also for your COMP2002 group projects. This will be great for your design blueprints in your reports.

**Aims:**

- Practice your object-oriented analysis and design/maintenance skills
- Practice working in small design teams
- Consider how to design software with lower maintenance effort in the future

PREPARATION AND POST PROCESSING

This lab session is a **group exercise**, and you are asked to work with your **COMP2002 group project team members** on this! The way you do it, is up to you. You can go for face-to-face meetings in A32, any other location, or use Teams and do it online (using your group project Teams site). In the latter case you should use a virtual whiteboard, so that you can draw diagrams together. If you have questions, ask the lab helpers if you are in Lab A32 or ask us on the Teams Questions channel, if you are elsewhere.

To set the scene, imagine you are competing in a "[Virtual UML Speed Refactor/Extend Challenge](#)". Here are the rules: Overall you have **120 minutes to fulfil the given task**. It is advised that you split up your precious time into two chunks: Use 90 minutes for the design and 30 minutes for summarising the outcome in form of a Power Point presentation. To get through the entire task in time you might want to have an initial discussion with the whole teams and then split up into smaller teams (perhaps pairs, if that fits) to do some 15-minute sprints. After each sprint you then reconvene briefly as a group and give some feedback to each other's outputs.

At the end of the lab session each group project team is welcome to submit a Power Point presentation (**one per group**) with a summary of the outputs you produced during the lab session (use case specification and screenshots / photos of UML diagrams). Your submissions will not be marked, but we are very interested to see what innovative and adventurous designs you are coming up with within the short time given. The submission link on Moodle will only be open until the end of the lab session. Please understand, that due to time constraints, we will not be able to provide individual feedback for each of the submissions.

You have been asked by the Student Union to develop a new "Student-Internship Match" application for them. A legacy system exists but it was abandoned some years ago. The idea is to match students with the right skillset to internship offers provided by local companies.



Before students and companies can use the application, they need to register and once they received their username/password they can log in. After registration students and companies need to add a profile which will be stored in a database (together with their login details). The profile can be edited at any time. Companies can then start posting job offers which will be stored in the database. Student profiles consist of student details and skills while job offers consist of company details, a job description, and requirements. Skills and requirements are to be submitted in form of predefined keywords so that they are easy to be matched in later searches.

Once the data has been stored in the database students as well as companies can then query the database and a query engine will try to match job offers with student profiles and vice versa. At a later stage the student union is planning to replace the keyword search with an intelligent query engine. It is a requirement that the application is designed following object-oriented principles and that it is easy to maintain and extend. If you have any ideas for extensions in order to improving the usability of the application, you are encouraged to add these to your design.

The legacy system was operated by a secretary who would handle all the data input and search requests. The only surviving design artefact from this system is the class diagram shown in Figure 1.

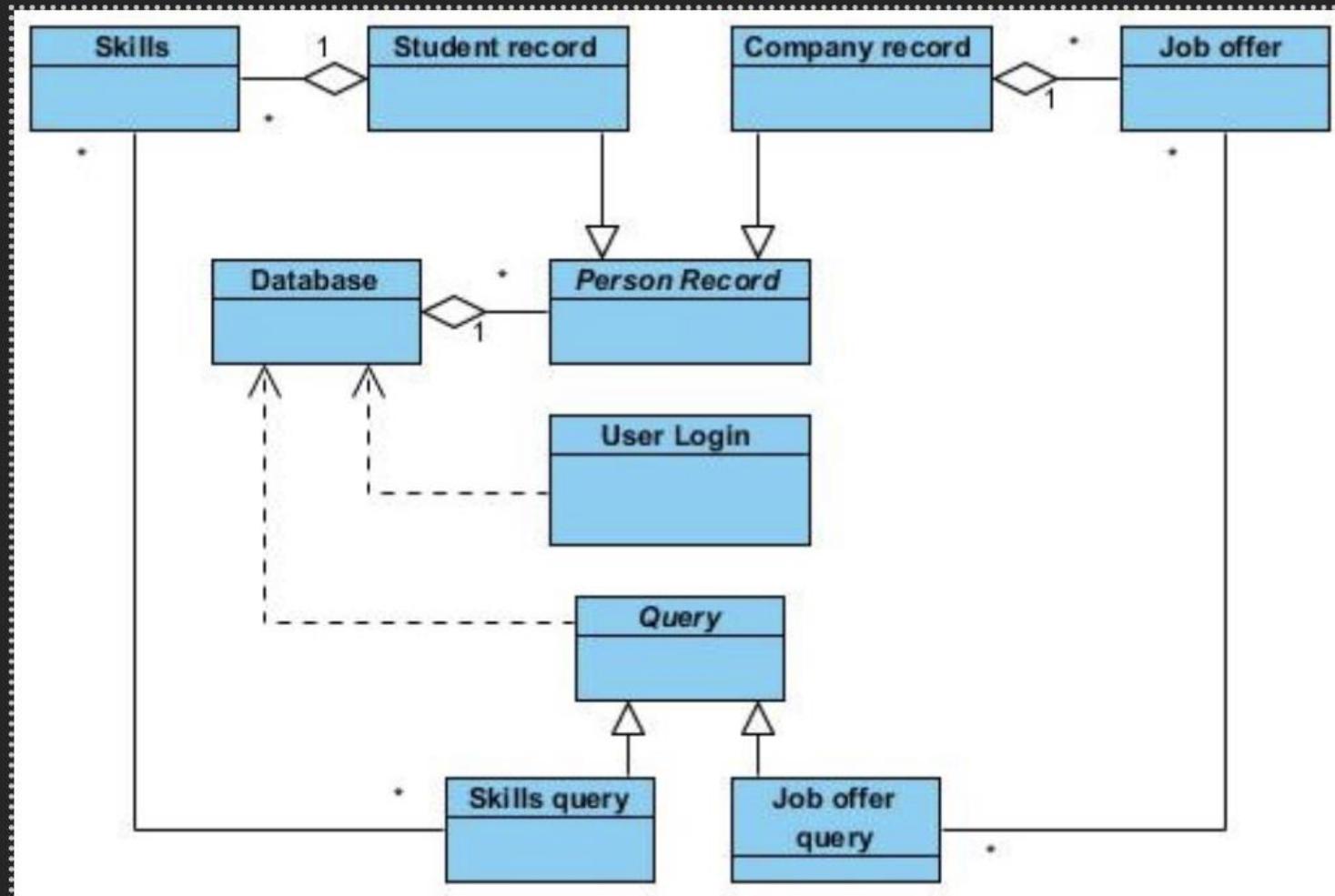


Figure 1: Legacy system class diagram

YOUR TASK

Find your **COMP2002 group project team members** and work on the object-oriented analysis and design for the refactoring and extension of the "Student-Internship Match" application.

You are asked to produce the following for the new version of the application:

- Use-case diagram for the described application
- Complete use case specification for a nontrivial use case (e.g. "search for jobs")
- Activity diagram of the same use case
- Sequence diagram of the same use case
- Class diagram
 - Classes including key attributes and operations
 - Relationships
 - Multiplicity indicators
- Nontrivial state machine diagram for one of the classes

SOME TIPS

In case you want to use Visual Paradigm for creating your diagrams, here are the links:

- Visual Paradigm Community Edition:
 - <https://www.visual-paradigm.com/download/community.jsp> <https://www.visual-paradigm.com/download/community.jsp>
- Visual Paradigm User Guide:
 - <https://www.visual-paradigm.com/support/documents/vpuserguide/>
- Visual Paradigm Online:
 - <https://online.visual-paradigm.com/diagrams/>
Scala Build Tool)

To improve maintainability in the future, think about these points when you design your system:

- Keep it as simple as possible (KISS principle)
- Keep similar functionality together, and different functionality apart (encapsulation)
 - This is also known as "high cohesion, and loose coupling"
- How easy would it be to change a module or feature in the future?
- How good are your diagrams at explaining the system to a new programmer?

</>

Coursework preview

COMP2013 Coursework Task Description

</>

This coursework is contributing 75% to your overall 20 CR COMP2013 assessment mark and it will be marked out of 100.

Recommendations: We recommend dedicating approximately 40-60 total hours on the coursework. We expect 30-40 hours of work for those who are aiming for a pass (40+) and 60 or more hours of work for those that are aiming for a first (70+). Please keep in mind that the skill level varies quite a bit in a class with over 350 students, and that the exact number of hours depends on your individual skill level. To help you, we have less required lectures in the last two weeks of term and some of the lab sessions are dedicated to your coursework.

Deadline:

- Milestone 1 – Friday the 24th of November 2023
- Milestone 2 – Tuesday 12th of December 2023

Friday 12/12/2022 @ 3pm (to be confirmed)

Assessment: The marks will be split as follows:

- 10% for git use (e.g. push, branch, merge, providing .gitignore)
- 30% for refactoring
- 30% for additions
- 10% for documentation (UML + Software Documentation (readme file + source code))
- 20% for the demonstration videos (Milestone 1 and 2), explaining your: git + documentation, maintenance activities and additions

Questions: Questions can be asked in person during the lab sessions and on Teams. Please read the questions that have already been asked on Teams before you ask or post yours, to avoid duplication. We will try to answer your questions as quickly as possible.

Requirement Specification - Overview

- Basic Software Maintenance on the provided code
- Better organisation of the code
- Excellent Design Documentation
- Extend the delivered code base by adding additional features
(some ideas will be provided, but additional features are always a bonus)
- Refactoring the code by adding design patterns (e.g. MVC), etc.
- Version Control with Git
- Testing Strategy
- Object Oriented Design and Implementation
- Improve legacy codebase
- Present your work well



Marking Scheme Overview

- Git (10%)
- REFACTORING (30%)
- ADDITIONS (30%)
- DOCUMENTATION (10%)
- DEMONSTRATION AND PRESENTATION OF THE WORK (20%)

Marking Scheme Overview

- Milestone 1
 - Understand the code base
 - Set up your code in your IDE
 - Set up Git
 - Initial Class Diagram (Refactoring)
- Milestone 2
 - Everything else

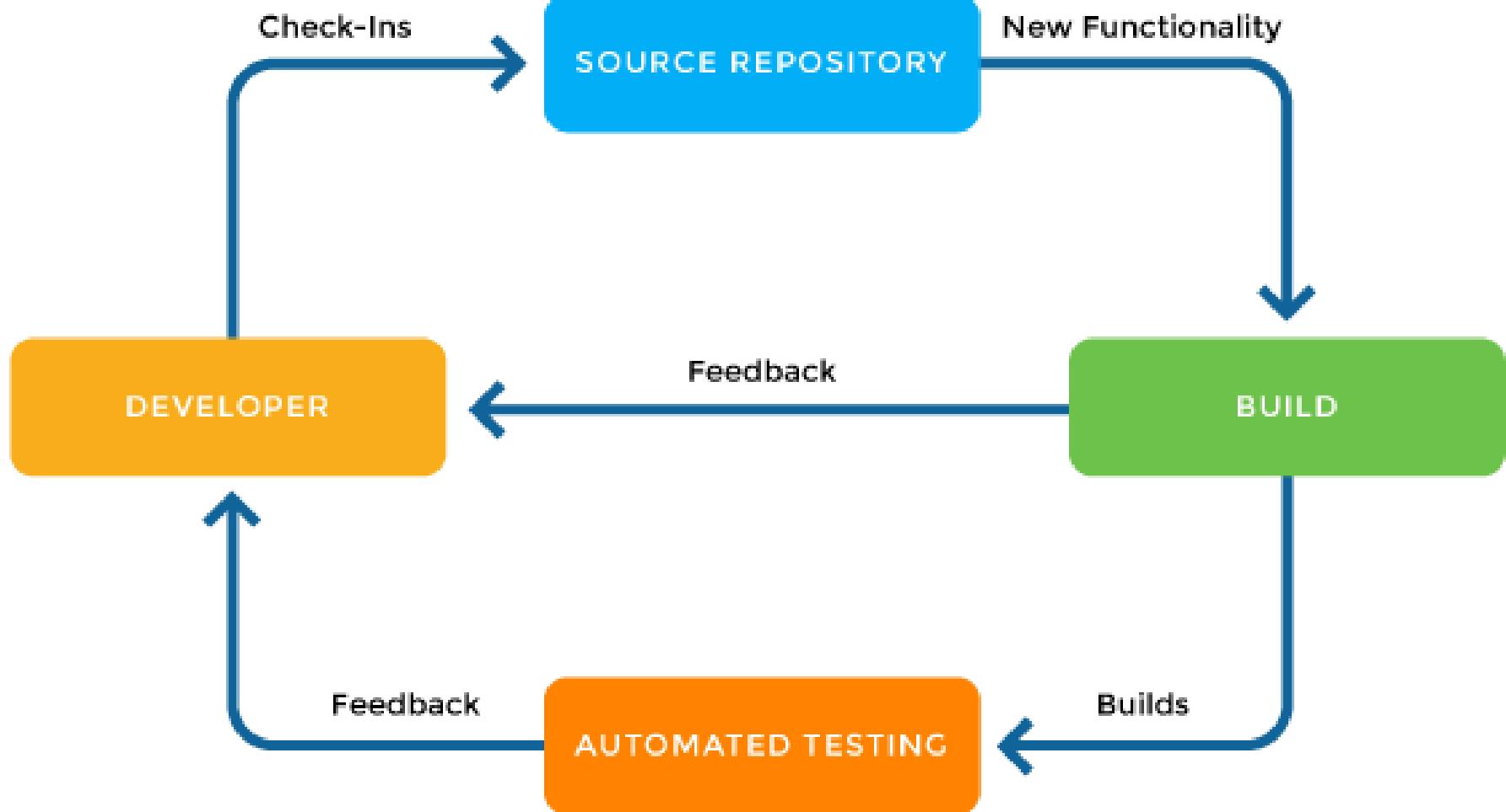
</>



University of
Nottingham
UK | CHINA | MALAYSIA

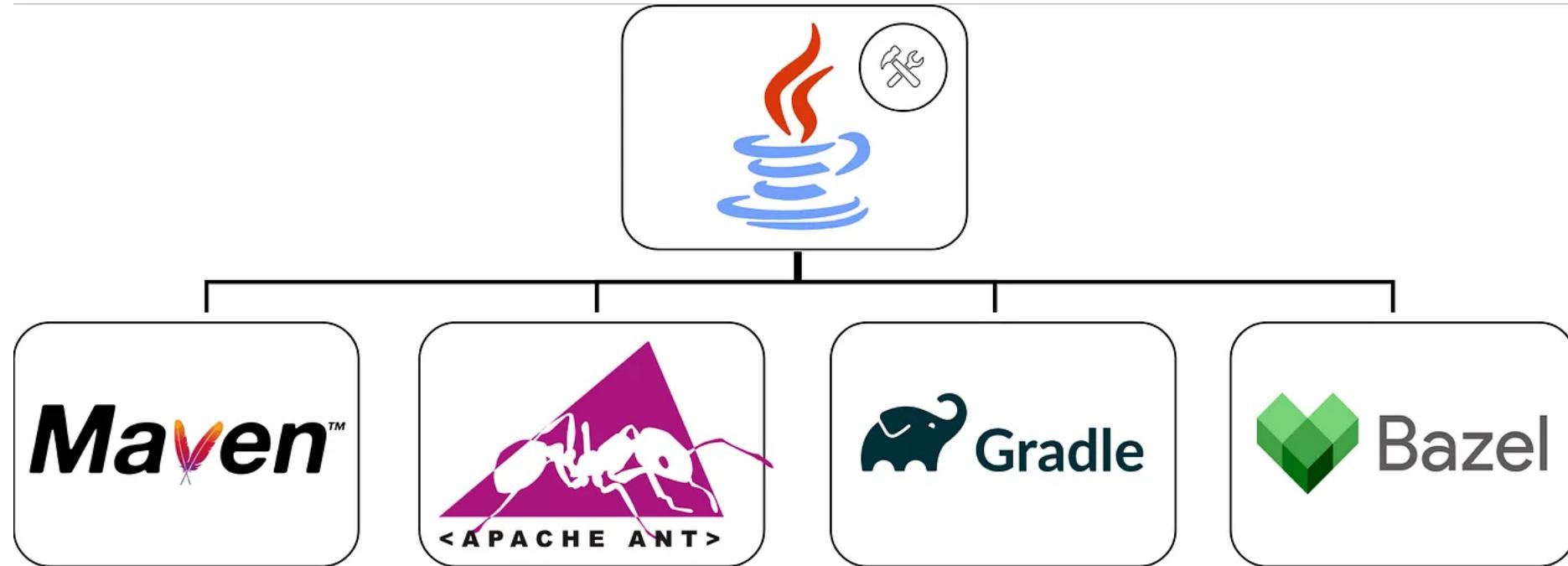
</>

Build Tools



Benefits of Build tools

- Project Compilation
- Reduces errors from manually running steps,
- Increases the consistency of the process.
- Dependencies
- Allow for better maintenance (keep a build log)
- Automated testing
- Deployment
- Supports documentation
- Repository management
- ...

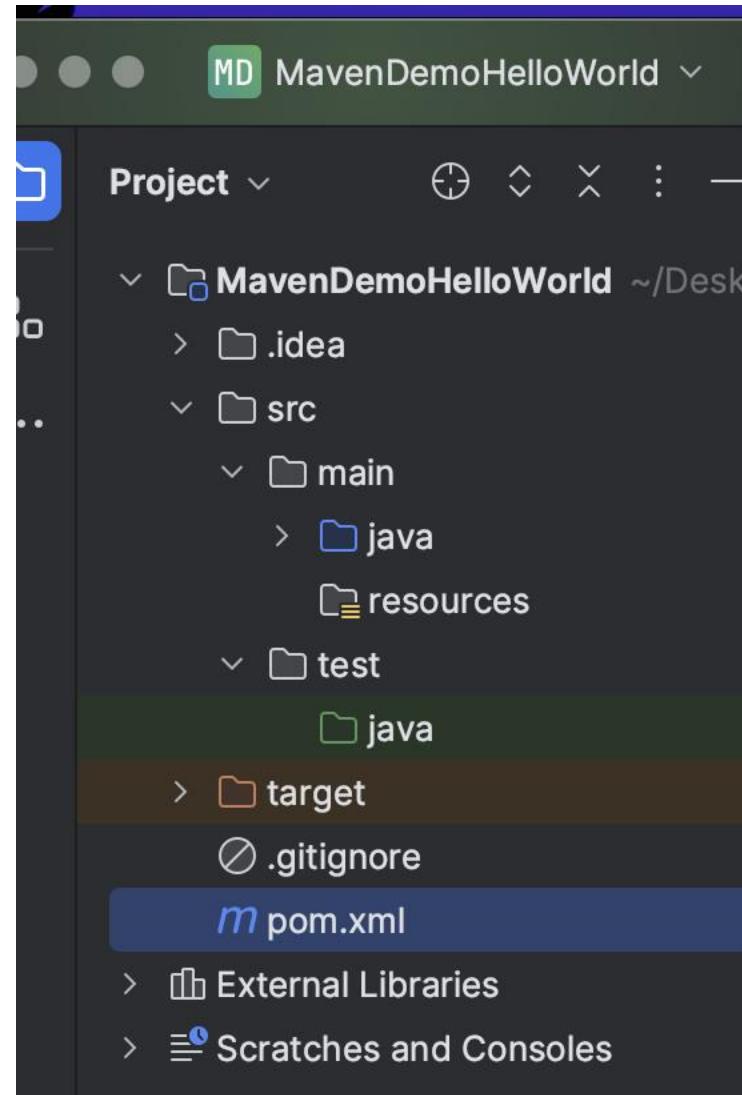


What is Maven?

- A build tool

What is Maven?

- A build tool
- Maven builds are structural

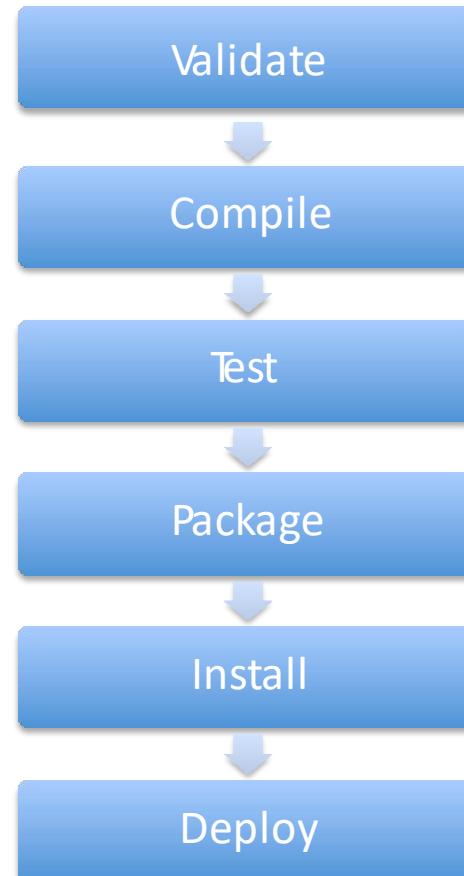


What is Maven?

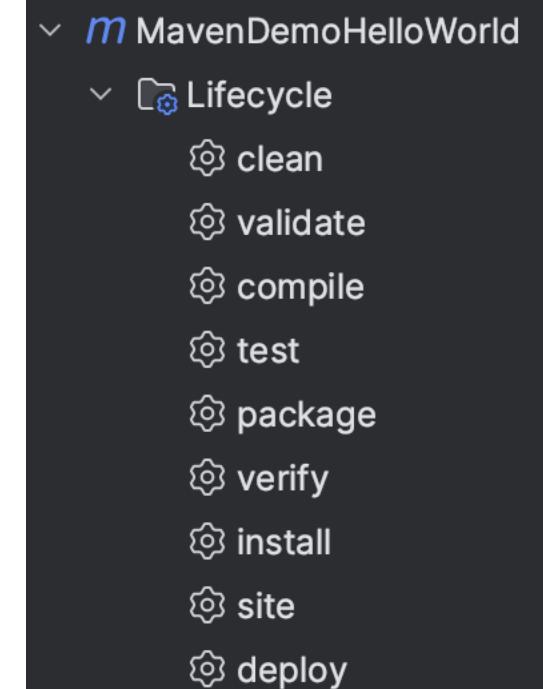
- A build tool
- Maven builds are structural
- Manages dependencies
 - You describe your dependencies, Maven downloads and includes all you need in your build path
- Manages repositories
 - Your projects share a common set of artifacts/jar files
 - No need to check in JAR files to version control

Maven Project Management

- Maven builds follows a lifecycle, using phases
 - Each lifecycle phase includes the ones above it
 - (Package starts with compile, then test, then package)
 - Plugins can customize or hook- in to the phases
 - Key Maven lifecycles
 - **Default** – your normal lifecycle
 - **Clean** – issued during the mvn cleanup command
 - **Site** – issued during the mvn site command



The Default Lifecycle



Maven Demo: Getting Started

What is a POM file?

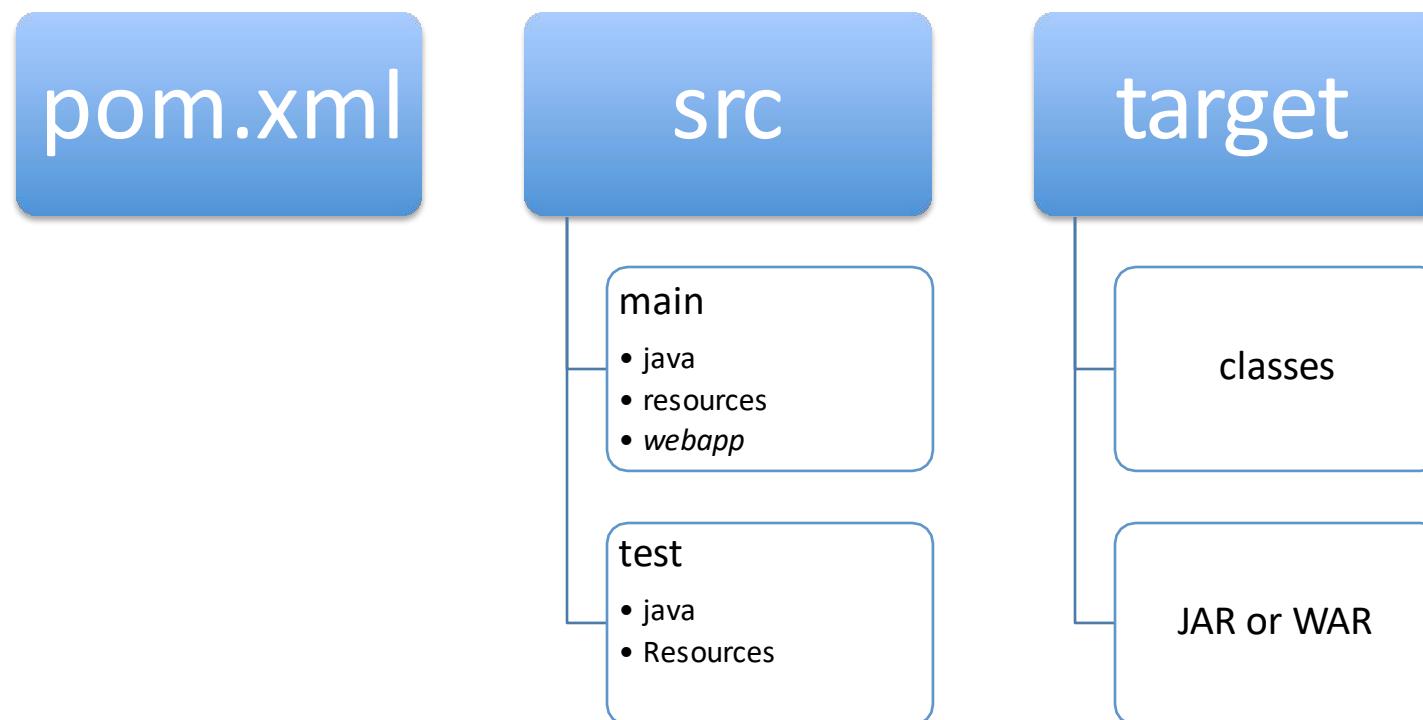
- Project Object Model (POM)
- Contains configuration information about your project
- Key items
 - Project Identification
 - Dependencies
 - Plug-Ins
 - Other Settings

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.COMP2013</groupId>
8   <artifactId>MavenDemoHelloWorld</artifactId>
9   <version>1.0-SNAPSHOT</version>
10
11  <properties>
12    <maven.compiler.source>20</maven.compiler.source>
13    <maven.compiler.target>20</maven.compiler.target>
14    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15  </properties>
16  <dependencies>
17    <dependency>
18      <groupId>junit</groupId>
19      <artifactId>junit</artifactId>
20      <version>4.13.2</version>
21      <scope>test</scope>
22    </dependency>
23  </dependencies>
24
25 </project>
```

What happens when you Set Up a Maven Project?

- Mavens build a project skeleton
 - Maven used its' conventions to place files in the right places
- Let's look at the directory structure...

Directory Structure, Basic Maven Project



Using Maven Commands

- The maven command (mvn)
 - mvn clean – removes files in target
 - mvn compile – compiles a project
 - mvn test – Runs all tests in the src/test directory
 - mvn package – builds the final target artifact (JAR, WAR)
 - mvn install – Installs the target artifact in your local repository
 - mvn deploy – Deploys the artifact (if configured)

Helpful Maven Reports

- **mvn site** -- Generates a web site report in target/site/index.html
- **mvn pmd:pmd** – runs a PMD (programming mistake detector) report, a source code analyzer for finding common flaws, output in target/site/pmd.html
- **mvn cobertura:cobertura** – runs a cobertura code test coverage report, the percentage of branches/lines accessed by unit tests, output in target/site/cobertura/index.html
- **mvn jdepend:generate** – runs a report on coupling between packages, output in target/site/jdepend-report.html
- **mvn checkstyle:checkstyle** – runs a checkstyle report
- **mvn javancss:javancss-report** – Runs a JavaNCSS (non commenting source statements) report to show method complexity, roughly equivalent to counting “;” and ‘{’ characters in Java source file
- Note: all of these and more can be configured in the <site> section of the maven build... YMMV.

Managing Dependencies

- Key fact: Maven projects have ONE Artifact (JAR, WAR)
 - Can build projects that depend on other projects
 - Can build parent projects that build child projects
 - Can depend on other open source libraries and frameworks
- Manage all of this via the <dependencies> tags

- Adds dependency to jUnit
 - Downloads 3.8.1 of Junit and stores in your maven repository
 - Only uses in path for testing

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Dependency Tips...

- Look for public dependencies on
www.mvnrepository.com
- Split out re-usable functionality into JAR projects
 - Create top-level projects with a “pom” target which coordinate the build of subordinate projects
 - Manage your own shared projects using a company repository (Archiva, others)

What we didn't cover...

- Transitive dependencies
 - If one dependency needs version A of a project, but another needs version B... You have to exclude the download of the wrong version
- We didn't cover multiple projects
- We didn't talk much about plugins
- Sites?
- But this will get you started

Resources

- Maven web site: <http://maven.apache.org>

Gradle: Next week



Acknowledgements

Thanks to:

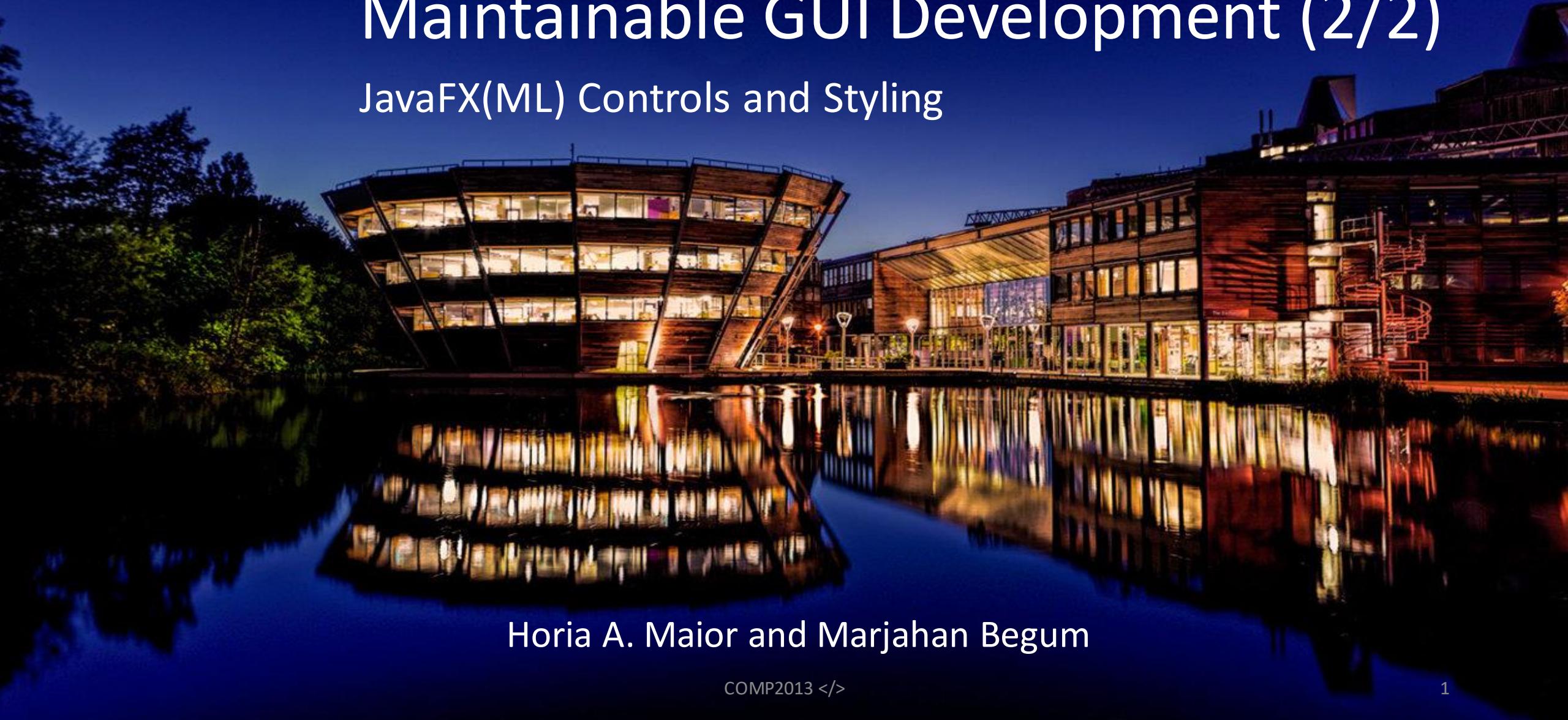
Robert Laramee, Ken Rimple of Chariot Solutions and Julie Greensmith for slides



Lecture 05A

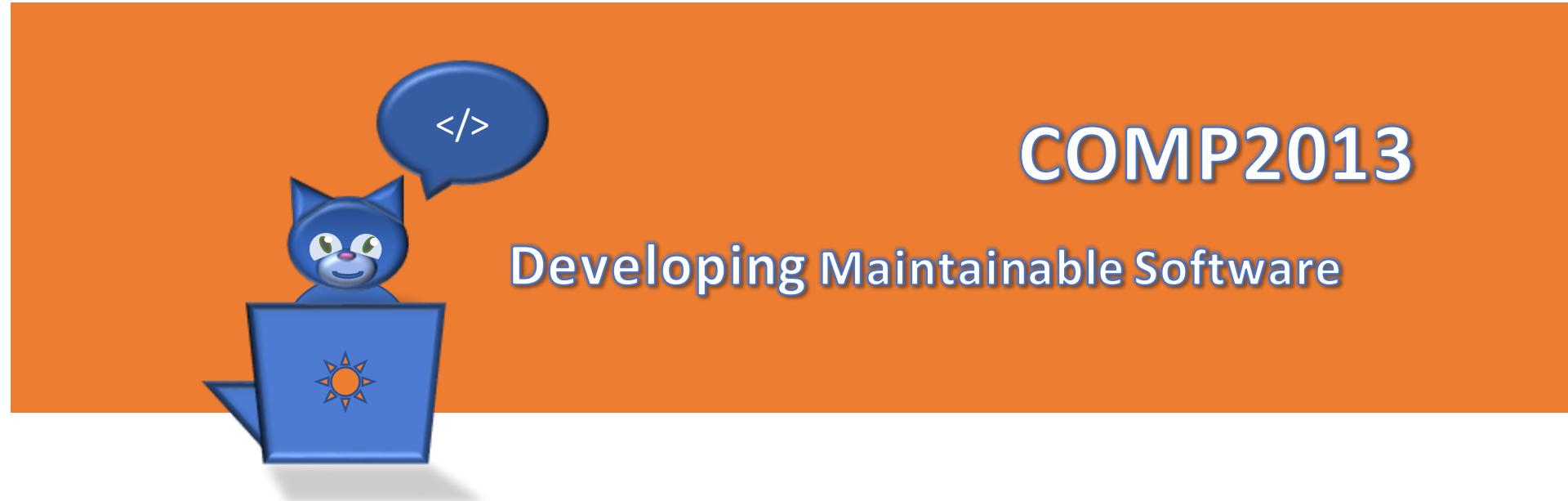
Maintainable GUI Development (2/2)

JavaFX(ML) Controls and Styling



Horia A. Maior and Marjahan Begum

</>



Lecture 08A

Maintainable GUI Development (2/2)

JavaFX(ML) Controls and Styling

Horia A. Maior and Marjahan Begum

Topics for this Week

</>

- Lecture 05A
 - Build Tools: Maven and Gradle (Not delivered last week)
 - JavaFX(ML) controls and styling
- Lab 05:
 - More complex GUI development
- Lecture 05B:
 - Coursework Q/A and support
 - Labsheet 5

</>

Coursework Support

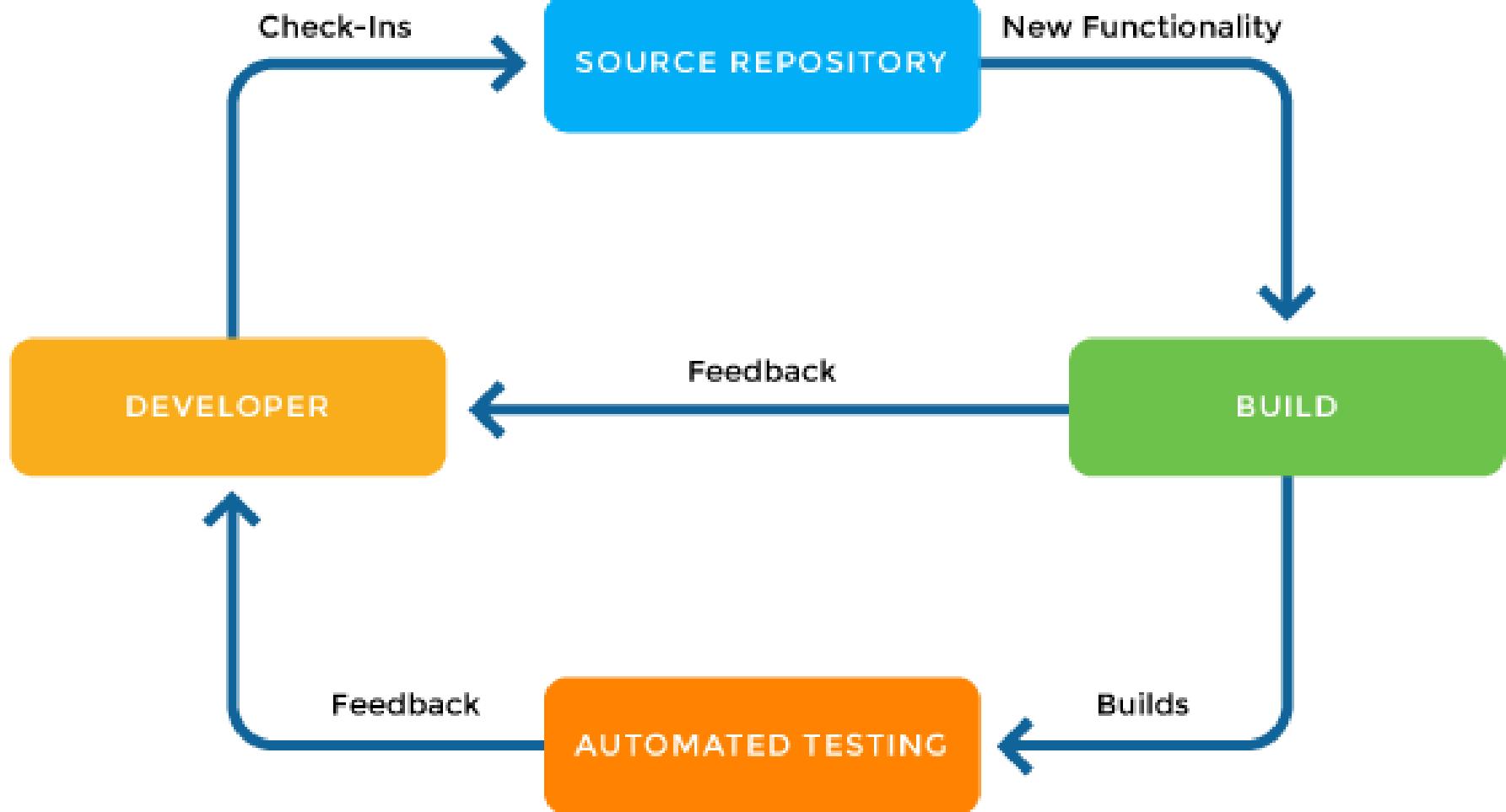
<https://forms.office.com/e/grwgXdn0YF>

COMP2013 Coursework Questions



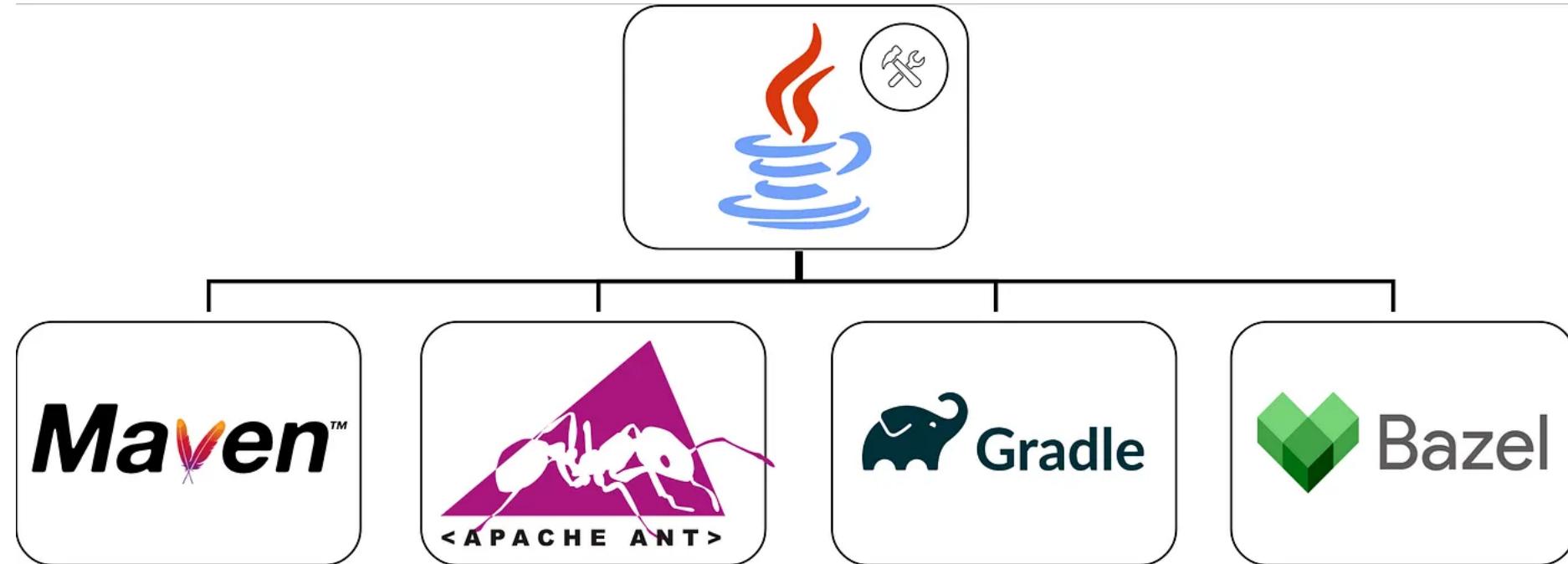
</>

Build Tools



Benefits of Build tools

- Project Compilation
- Reduces errors from manually running steps,
- Increases the consistency of the process.
- Dependencies
- Allow for better maintenance (keep a build log)
- Automated testing
- Deployment
- Supports documentation
- Repository management
- ...

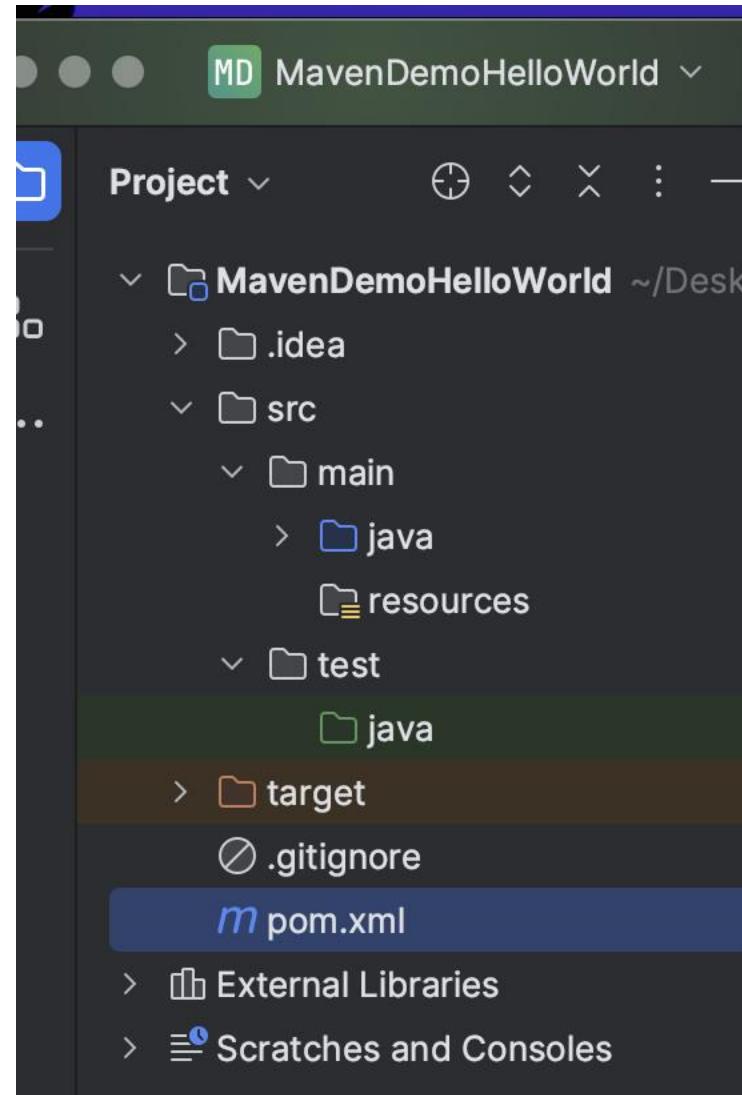


What is Maven?

- A build tool

What is Maven?

- A build tool
- Maven builds are structural

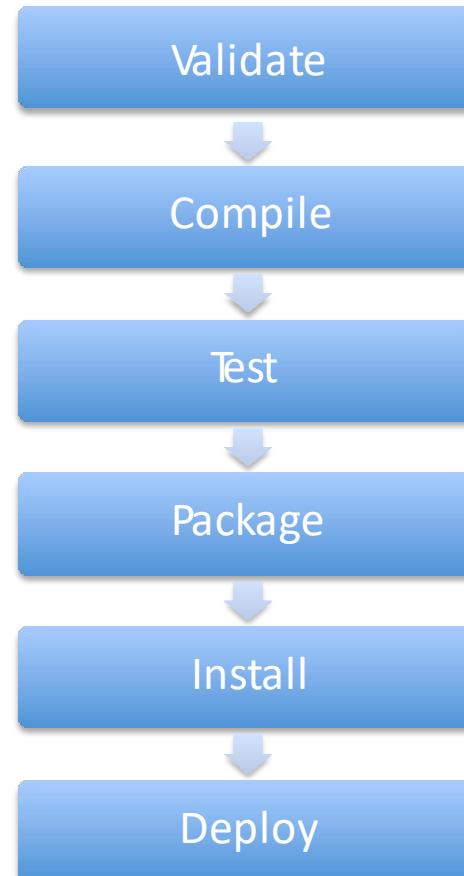


What is Maven?

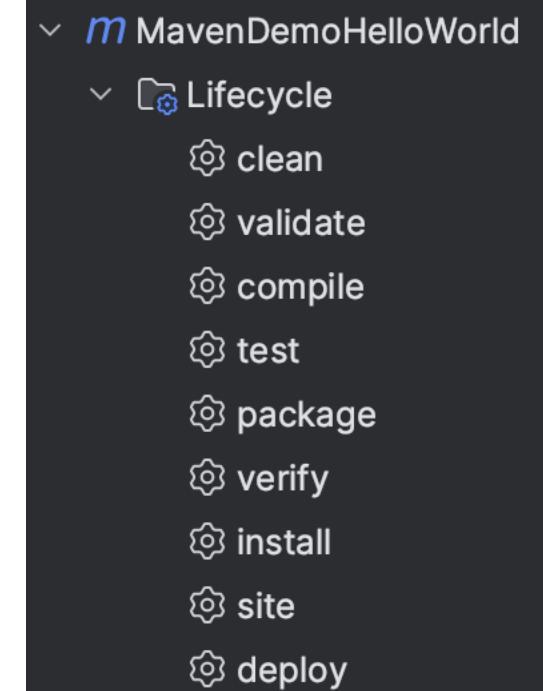
- A build tool
- Maven builds are structural
- Manages dependencies
 - You describe your dependencies, Maven downloads and includes all you need in your build path
- Manages repositories
 - Your projects share a common set of artifacts/jar files
 - No need to check in JAR files to version control

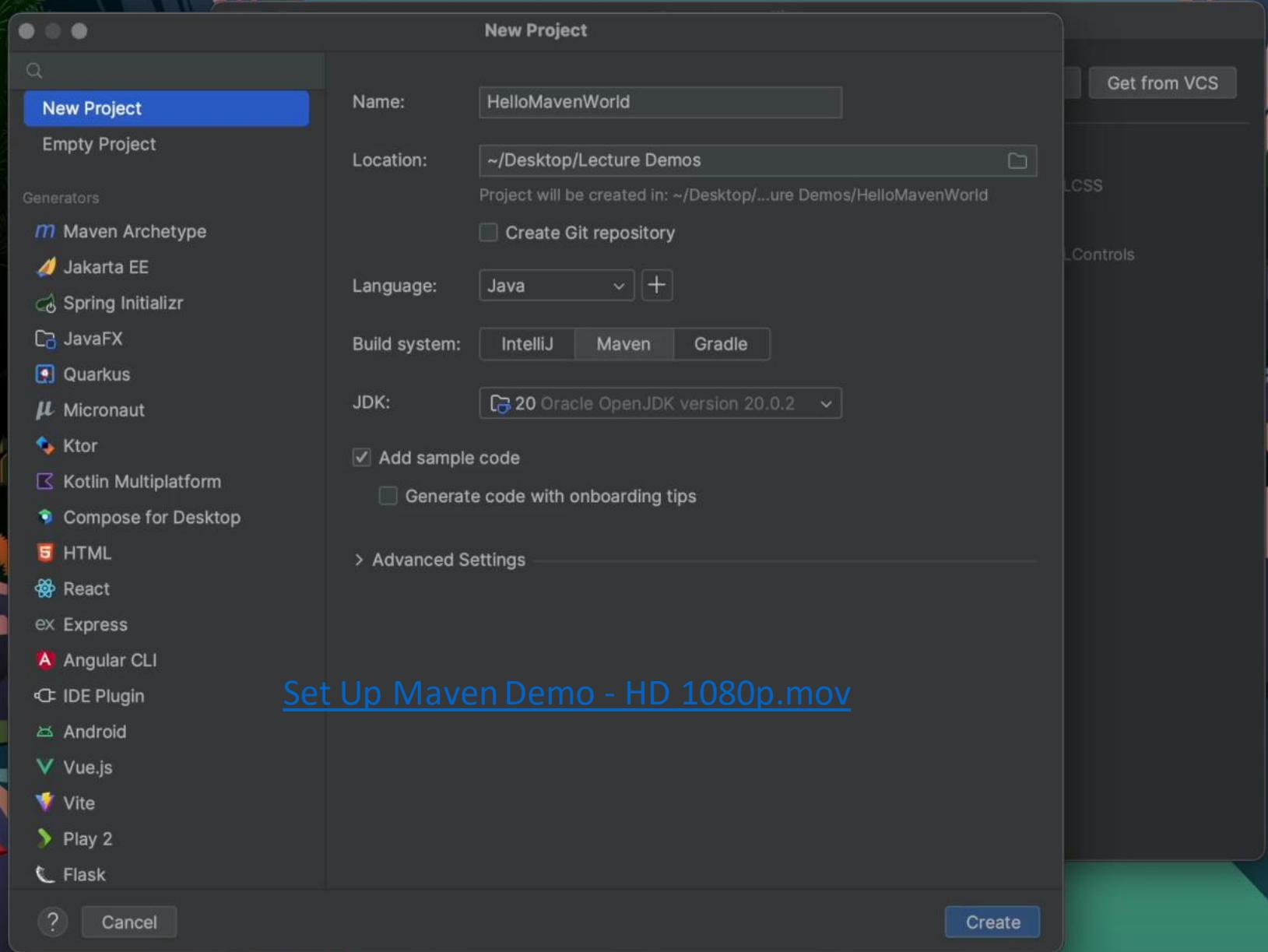
Maven Project Management

- Maven builds follows a lifecycle, using phases
 - Each lifecycle phase includes the ones above it
 - (Package starts with compile, then test, then package)
 - Plugins can customize or hook- in to the phases
 - Key Maven lifecycles
 - **Default** – your normal lifecycle
 - **Clean** – issued during the mvn cleanup command
 - **Site** – issued during the mvn site command



The Default Lifecycle





What is a POM file?

- Project Object Model (POM)
- Contains configuration information about your project
- Key items
 - Project Identification
 - Dependencies
 - Plug-Ins
 - Other Settings

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.COMP2013</groupId>
8   <artifactId>MavenDemoHelloWorld</artifactId>
9   <version>1.0-SNAPSHOT</version>
10
11  <properties>
12    <maven.compiler.source>20</maven.compiler.source>
13    <maven.compiler.target>20</maven.compiler.target>
14    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15  </properties>
16  <dependencies>
17    <dependency>
18      <groupId>junit</groupId>
19      <artifactId>junit</artifactId>
20      <version>4.13.2</version>
21      <scope>test</scope>
22    </dependency>
23  </dependencies>
24
25 </project>
```

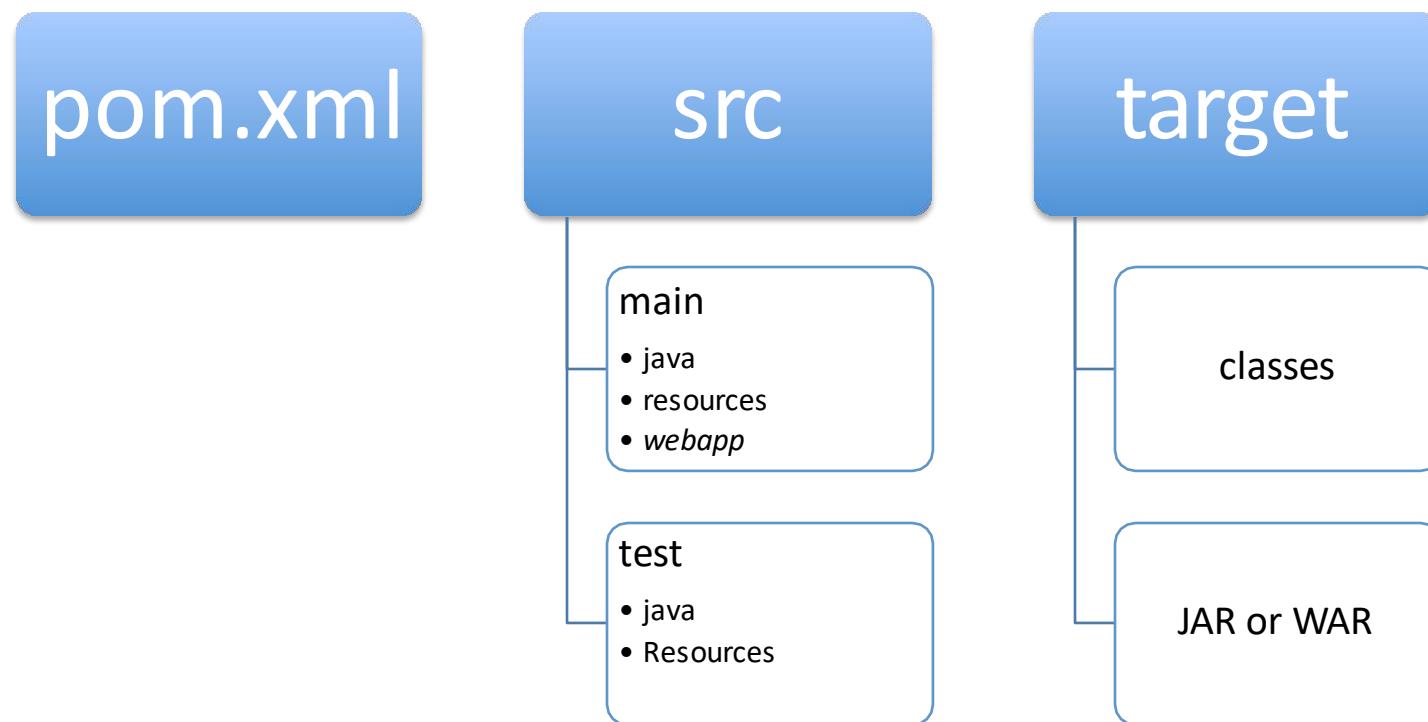
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach
5 <modelVersion>4.0.0</modelVersion>
6
7 <groupId>com.COMP2013</groupId>
8 <artifactId>MavenDemoHelloWorld</artifactId>
9 <version>1.0-SNAPSHOT</version>
10
11 <properties>
12   <maven.compiler.source>20</maven.compiler.source>
13   <maven.compiler.target>20</maven.compiler.target>
14   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15 </properties>
16 <dependencies>
```

```
8 <artifactId>havenDemoneetwerk</artifactId>
9 <version>1.0-SNAPSHOT</version>
10
11 <properties>
12     <maven.compiler.source>20</maven.compiler.source>
13     <maven.compiler.target>20</maven.compiler.target>
14     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15 </properties>
16 <dependencies>
17     <dependency>
18         <groupId>junit</groupId>
19         <artifactId>junit</artifactId>
20         <version>4.13.2</version>
21         <scope>test</scope>
22     </dependency>
23 </dependencies>
24
25 </project>
```

What happens when you Set Up a Maven Project?

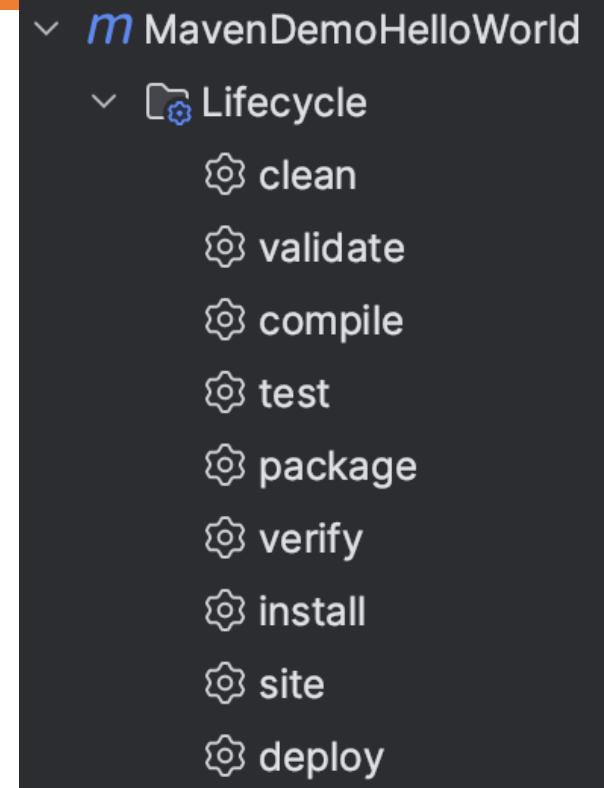
- Maven builds a project skeleton
 - Maven used its' conventions to place files in the right places
- Let's look at the directory structure...

Directory Structure, Basic Maven Project



Using Maven Commands

- The maven command (mvn)
 - mvn clean – removes files in target
 - mvn compile – compiles a project
 - mvn test – Runs all tests in the src/test directory
 - mvn package – builds the final target artifact (JAR, WAR)
 - mvn install – Installs the target artifact in your local repository
 - mvn deploy – Deploys the artifact (if configured)



Helpful Maven Reports

- **mvn site** -- Generates a web site report in target/site/index.html
- **mvn pmd:pmd** – runs a PMD (programming mistake detector) report, a source code analyzer for finding common flaws, output in target/site/pmd.html
- **mvn cobertura:cobertura** – runs a cobertura code test coverage report, the percentage of branches/lines accessed by unit tests, output in target/site/cobertura/index.html
- **mvn jdepend:generate** – runs a report on coupling between packages, output in target/site/jdepend-report.html
- **mvn checkstyle:checkstyle** – runs a checkstyle report
- **mvn javancss:javancss-report** – Runs a JavaNCSS (non commenting source statements) report to show method complexity, roughly equivalent to counting “;” and ‘{’ characters in Java source file
- Note: all of these and more can be configured in the <site> section of the maven build... YMMV.

Managing Dependencies

- Key fact: Maven projects have ONE Artifact (JAR, WAR)
 - Can build projects that depend on other projects
 - Can build parent projects that build child projects
 - Can depend on other open source libraries and frameworks
- Manage all of this via the <dependencies> tags

- Adds dependency to jUnit
 - Downloads 3.8.1 of Junit and stores in your maven repository

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Dependency Tips...

- Look for public dependencies on
www.mvnrepository.com
- Split out re-usable functionality into JAR projects
 - Create top-level projects with a “pom” target which coordinate the build of subordinate projects
 - Manage your own shared projects using a company repository (Archiva, others)

What we didn't cover...

Resources

- Maven web site: <http://maven.apache.org>

Gradle

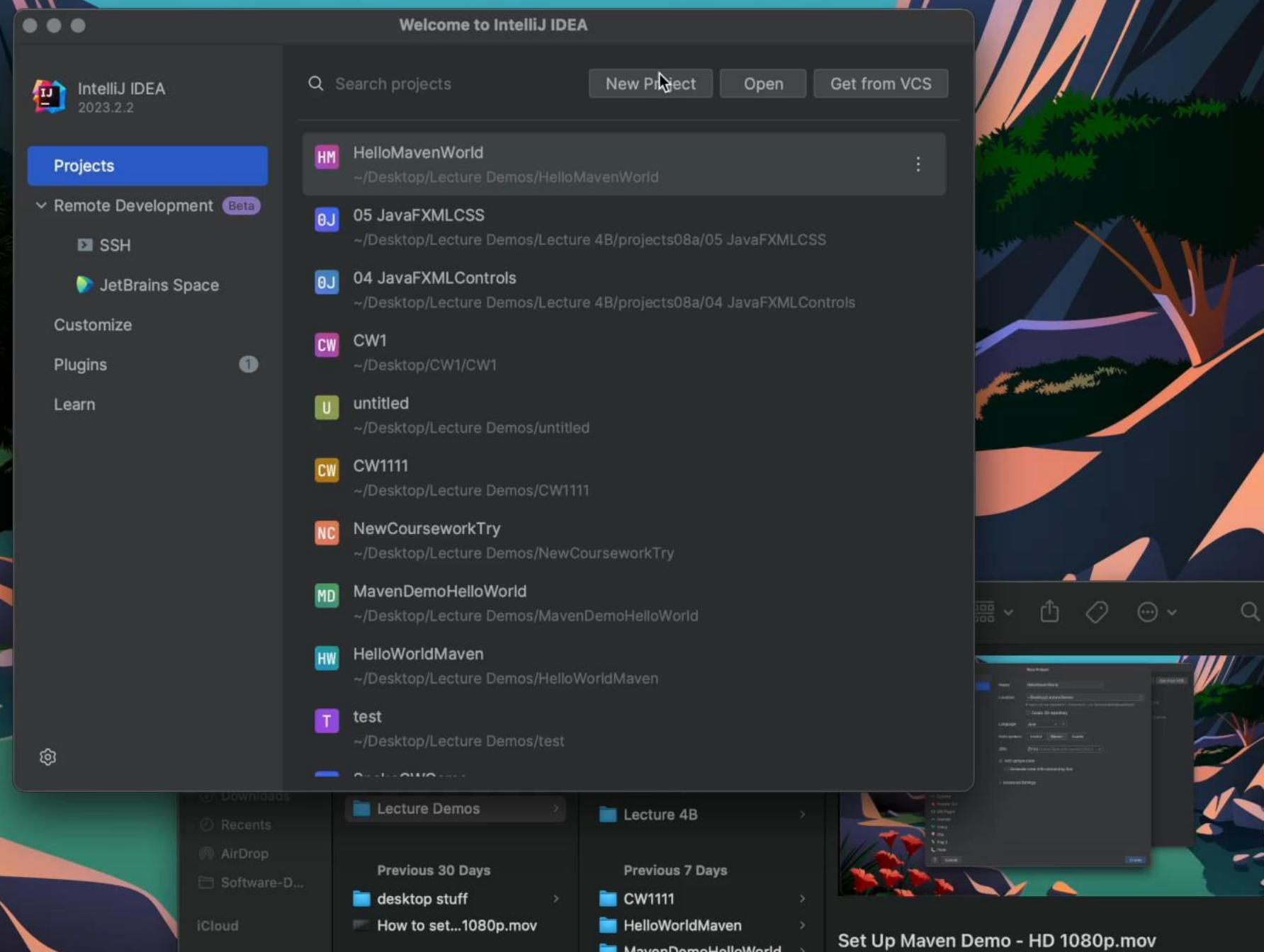


Gradle: A Contemporary Approach



- Replaces XML in favour of a domain specific language ‘groovy’
- Based on a programming language therefore can implement control flow
- Although it uses groovy itself, it can act as a build manager for any language
- Declarative: plug-ins add functionality
- More cleanly accomplishes required tasks of a typical development project, from compilation through testing and deployment.
- It’s the official build system for Android

Set Up Gradle - HD 1080p.mov



Grade Build Cycle

- It launches as a new JVM process
 - It parses the *gradle.properties* file and configures Gradle accordingly
- Next, it creates a Settings instance for the build
- Then, it evaluates the *settings.gradle* file against the Settings object
- It creates a hierarchy of Projects, based on the configured Settings object
- Finally, it executes each *build.gradle* file against its project
 - These files must be precisely named
 - Automatic setup using a **Grade Wrapper**

Build: Tutorial Videos

- Tutorial series on Gradle
 - Designed for the more advanced of you
- Practical tutorials, running time about an hour (plus additional videos)
- Video series:
 - Introduction to Gradle (38 minutes)
 - Gradle Introduction on the Virtual Pair Programmers YT channel
 - <https://www.youtube.com/watch?v=mPpncYETnTg&t=1021s>

Reading Homework Comparing Build File Tools

- Interesting blog articles for you to read comparing Maven vs Gradle:
 - [http://www.adam-bien.com/roller/abien/entry/maven vs ant](http://www.adam-bien.com/roller/abien/entry/maven_vs_ant)
 - <https://www.baeldung.com/ant-maven-gradle>

Acknowledgements

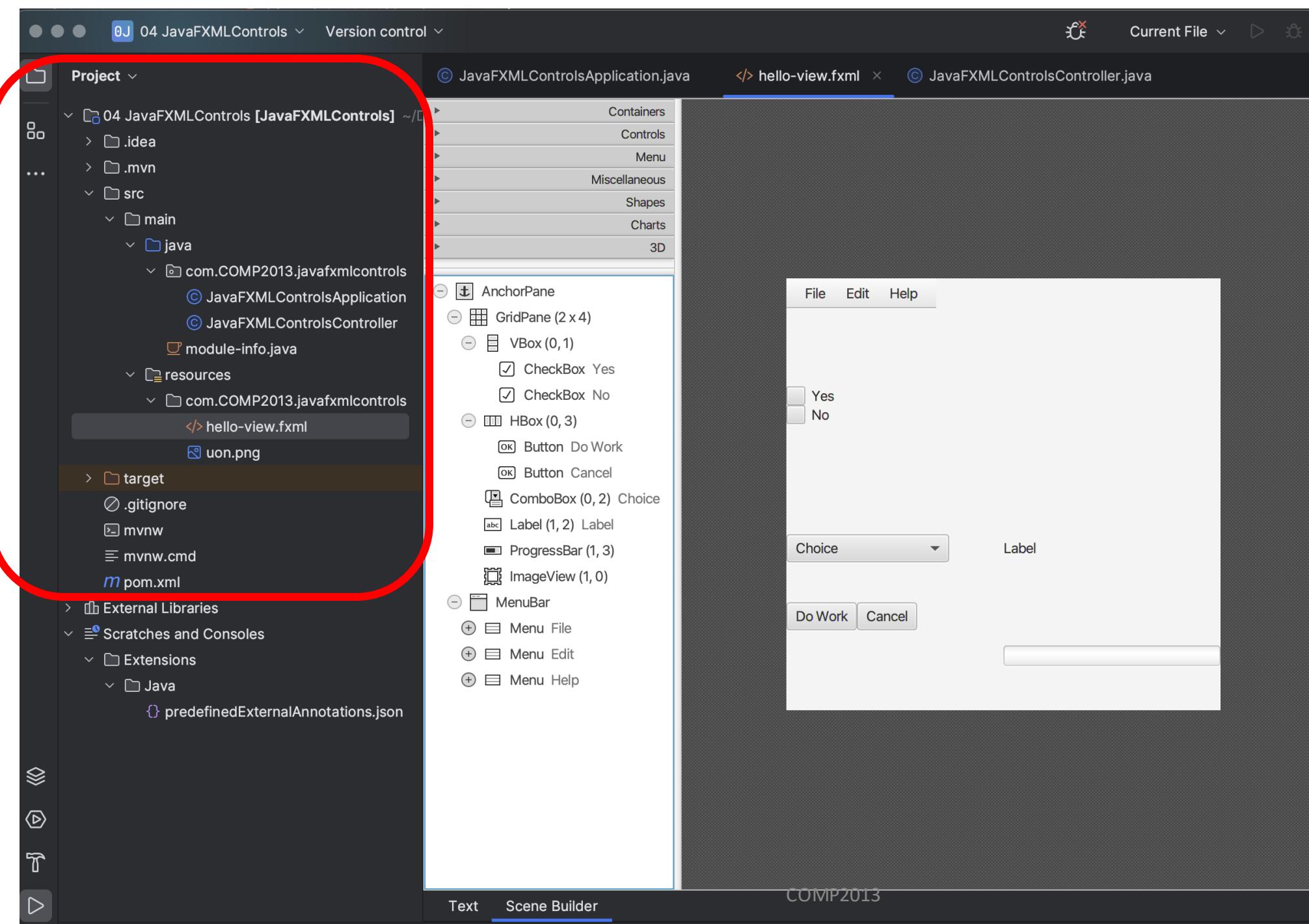
Thanks to:

Robert Laramee, Ken Rimple of Chariot Solutions and Julie Greensmith for the help with materials for this.

JavaFX advanced topics

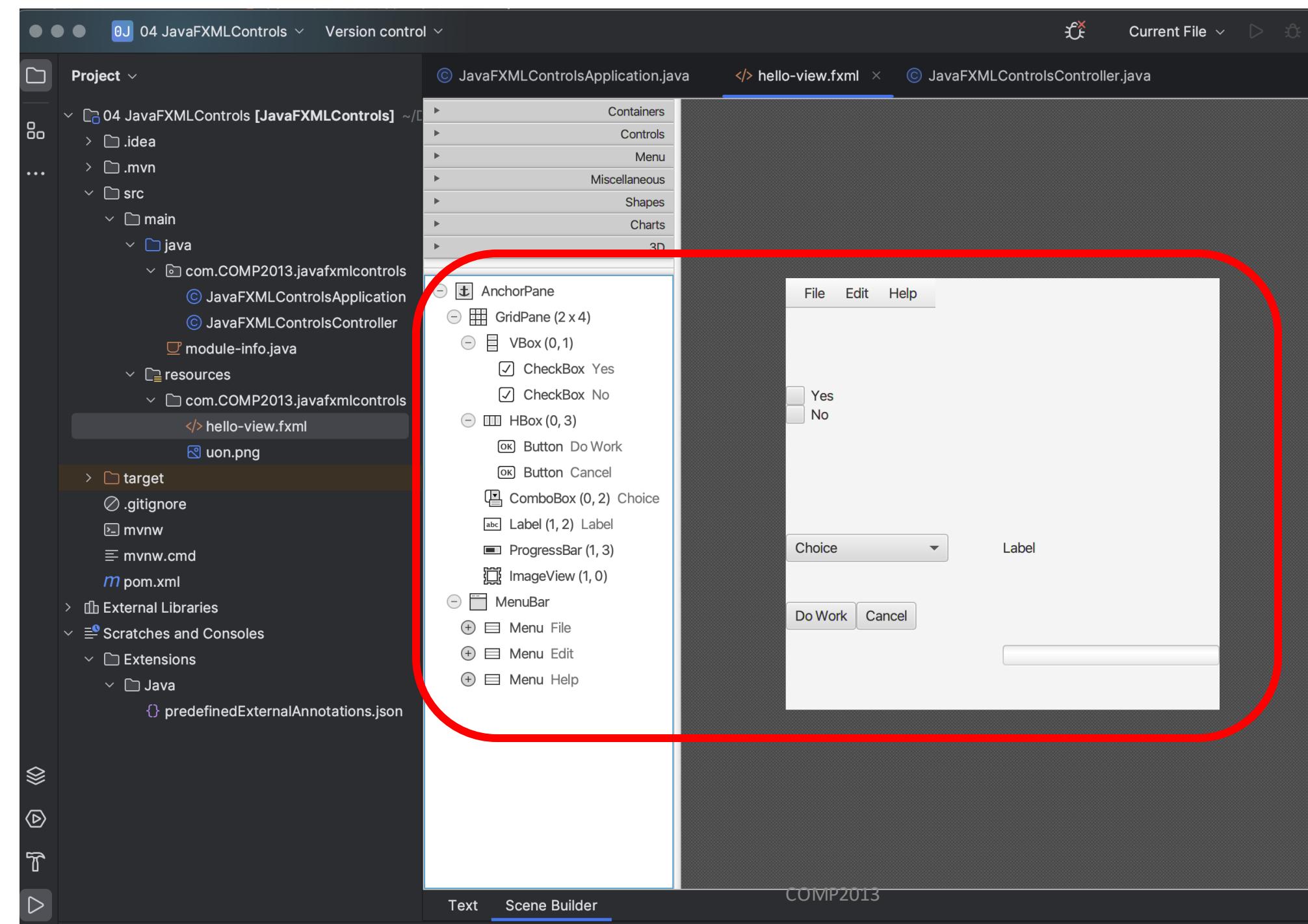
FXML

Preparation



</>

DEMO



Preparation

</>

- The result



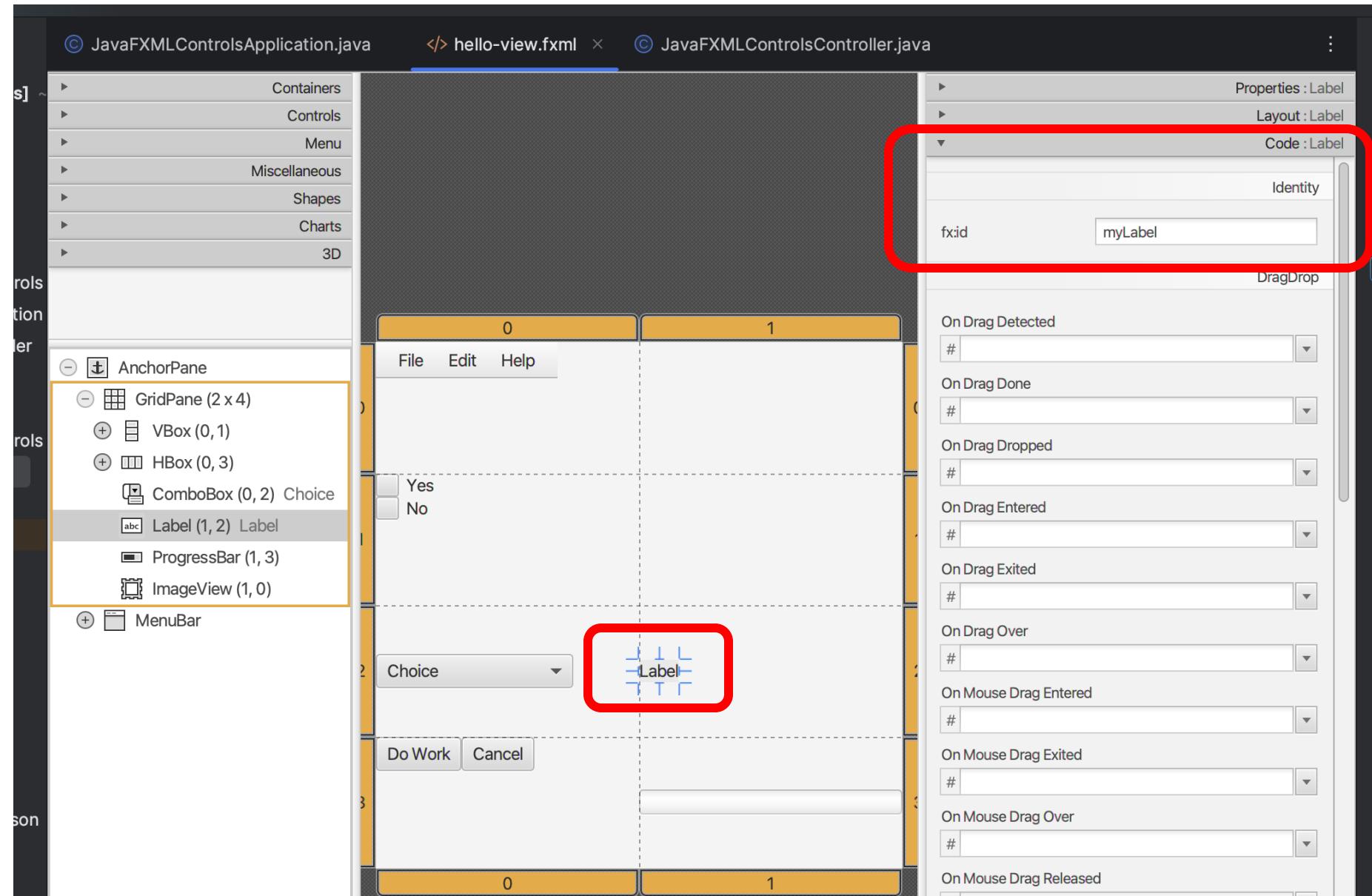
FXML

Controls

A selection of controls ... in more detail

</>

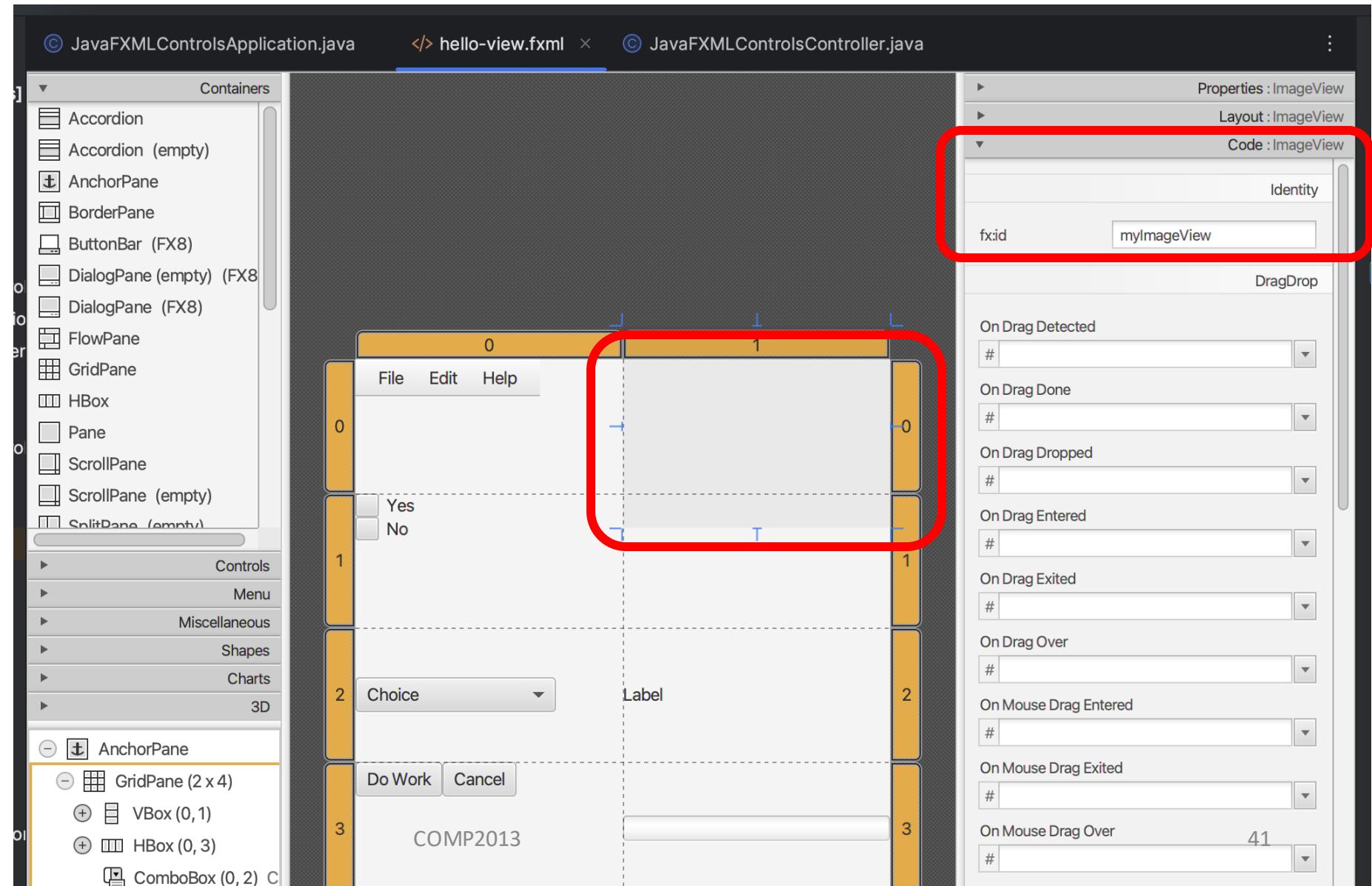
- My selection:
 - Label



A selection of controls ... in more detail

</>

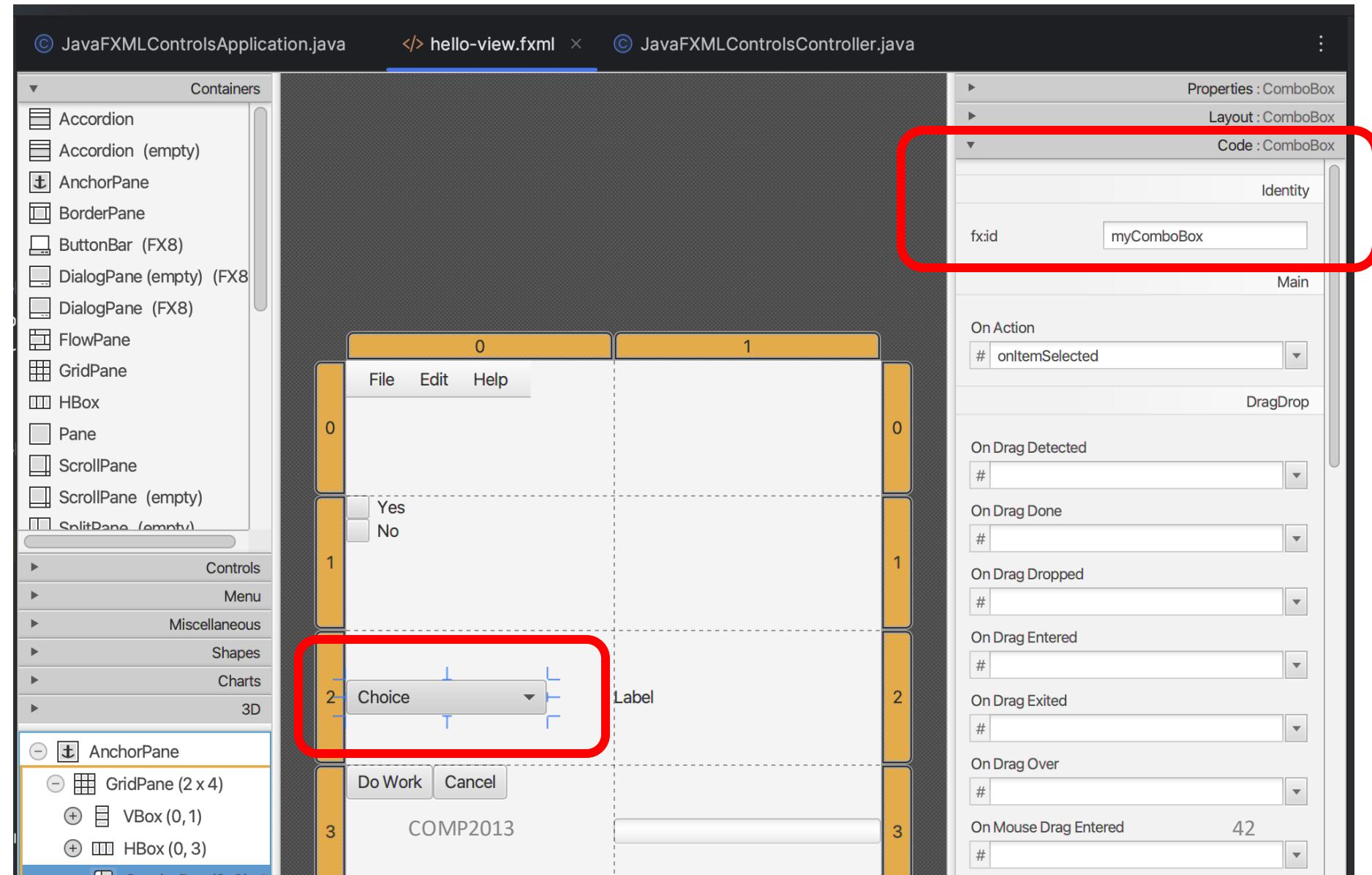
- My selection:
 - Label
 - ImageView



A selection of controls ... in more detail

</>

- My selection:
 - Label
 - ImageView
 - ComboBox



© JavaFXMLControlsApplication.java × </> hello-view.fxml © JavaFXMLControlsController.java

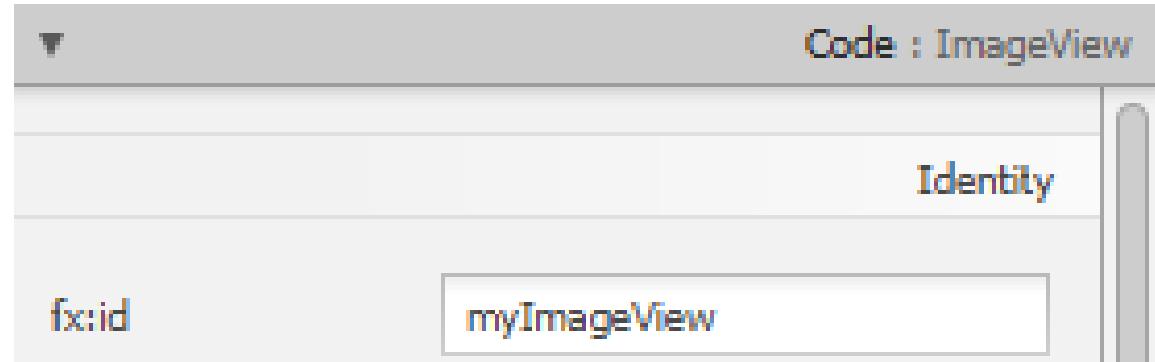
```
1 package com.COMP2013.javafxmlcontrols;
2
3 > import ...
4
5
6 D public class JavaFXMLControlsApplication extends Application {
7     @Override
8     public void start(Stage stage) throws IOException {
9         FXMLLoader fxmlLoader = new FXMLLoader(JavaFXMLControlsApplication.class.getResource( name: "hello-view.fxml"));
10        Scene scene = new Scene(fxmlLoader.load());
11        stage.setTitle("Hello!");
12        stage.setScene(scene);
13        stage.show();
14    }
15
16
17
18
19
20 D >     public static void main(String[] args) { launch(); }
21
22
23 }
```

```
3     > import ...  
12  
13     public class JavaFXMLControlsController {  
14  
15     16 </> @FXML private ComboBox<String> myComboBox;  
17 </> @FXML private ImageView myImageView;  
18 </> @FXML private Label myLabel;  
19 </> @FXML private CheckBox myNoCheckBox;  
20 </> @FXML private CheckBox myYesCheckBox;  
21  
22     @FXML  
23     private void initialize(){  
24         myYesCheckBox.setSelected(true);  
25         ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full","Half","Empty");  
26         myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());  
27         myComboBox.setItems(options);  
28         Image image=new Image(String.valueOf(JavaFXMLControlsApplication.class.getResource( name: "uon.png")));  
29         myImageView.setImage(image);  
30     }  
}
```

ImageView

</>

- Can be done with an ImageView Control
 - Give it an fx:id
 - Then in your code:



```
3    > import ...  
12  
13    public class JavaFXMLControlsController {  
14  
15  
16    </> @FXML private ComboBox<String> myComboBox;  
17    </> @FXML private ImageView myImageView;  
18    </> @FXML private Label myLabel;  
19    </> @FXML private CheckBox myNoCheckBox;  
20    </> @FXML private CheckBox myYesCheckBox;  
21  
22    @FXML  
23    private void initialize(){  
24        myYesCheckBox.setSelected(true);  
25        ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full","Half","Empty");  
26        myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());  
27        myComboBox.setItems(options);  
28        Image image=new Image(String.valueOf(JavaFXMLControlsApplication.class.getResource( name: "uon.png")));  
29        myImageView.setImage(image);  
30    }  
}
```

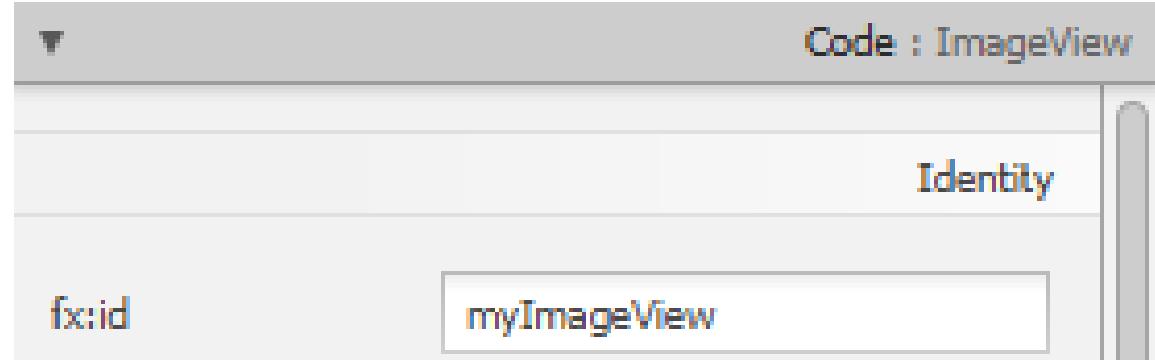
```
3     > import ...  
12  
13     public class JavaFXMLControlsController {  
14  
15  
16     </> @FXML private ComboBox<String> myComboBox;  
17     </> @FXML private ImageView myImageView;  
18     </> @FXML private Label myLabel;  
19     </> @FXML private CheckBox myNoCheckBox;  
20     </> @FXML private CheckBox myYesCheckBox;  
21  
22     @FXML  
23     private void initialize(){  
24         myYesCheckBox.setSelected(true);  
25         ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full","Half","Empty");  
26         myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());  
27         myComboBox.setItems(options);  
28         Image image=new Image(String.valueOf(JavaFXMLControlsApplication.class.getResource( name: "uon.png")));  
29         myImageView.setImage(image);  
30     }  
}
```

ImageView

</>

- Can be done with an ImageView Control

- Give it an fx:id
 - Then in your code:

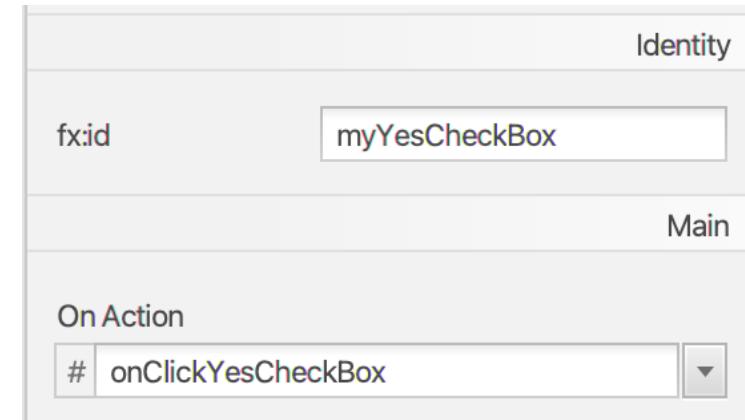
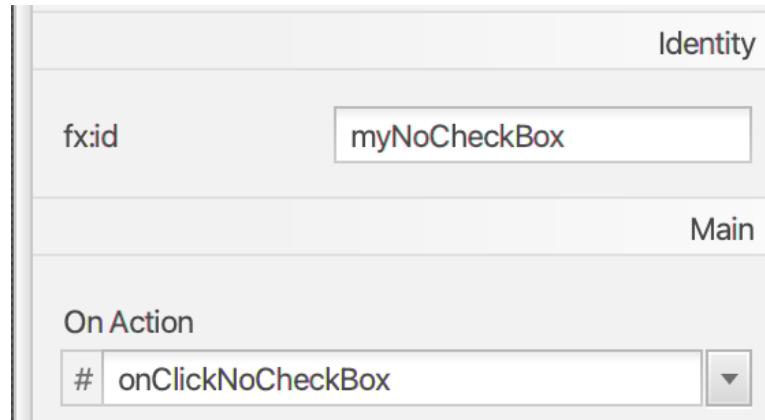


- Watch for working directories when loading files
 - My default is project\resources

CheckBox

</>

- Check boxes have a boolean state, depending whether checked or not
- Steps
 - Add your "onClickYesCheckBox()" method to the corresponding Controller class
 - Give it an fx:id and listen for the action



```
3     > import ...  
12  
13     public class JavaFXMLControlsController {  
14  
15  
16     </> @FXML private ComboBox<String> myComboBox;  
17     </> @FXML private ImageView myImageView;  
18     </> @FXML private Label myLabel;  
19     </> @FXML private CheckBox myNoCheckBox;  
20     </> @FXML private CheckBox myYesCheckBox;  
21  
22     @FXML  
23     private void initialize(){  
24         myYesCheckBox.setSelected(true);  
25         ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full","Half","Empty");  
26         myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());  
27         myComboBox.setItems(options);  
28         Image image=new Image(String.valueOf(JavaFXMLControlsApplication.class.getResource( name: "uon.png")));  
29         myImageView.setImage(image);  
30     }
```

```
32 @FXML  
33     private void onClickYesCheckBox() {  
34         if(myYesCheckBox.isSelected()){  
35             myNoCheckBox.setSelected(false);  
36         }else{  
37             myNoCheckBox.setSelected(true);  
38         }  
39     }
```

```
41 @FXML  
42     private void onClickNoCheckBox() {  
43         if(myNoCheckBox.isSelected()){  
44             myYesCheckBox.setSelected(false);  
45         }else{  
46             myYesCheckBox.setSelected(true);  
47         }  
48     }
```

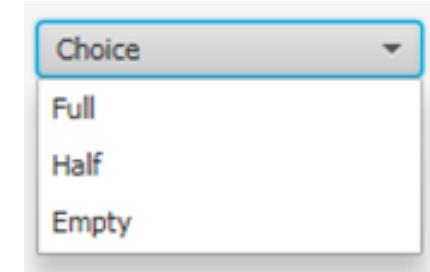
- The result



ComboBox

</>

- This is the 'drop down box' style of control.
- It presents the user with a list of items, and the user can select one.
- Slightly different to other controls so far:
 - Need to get items into the list
 - Need to check when an item had been selected



```
3     > import ...  
12  
13     public class JavaFXMLControlsController {  
14  
15  
16 </>     @FXML private ComboBox<String> myComboBox;  
17 </>     @FXML private ImageView myImageView;  
18 </>     @FXML private Label myLabel;  
19 </>     @FXML private CheckBox myNoCheckBox;  
20 </>     @FXML private CheckBox myYesCheckBox;  
21  
22     @FXML  
23     private void initialize(){  
24         myYesCheckBox.setSelected(true);  
25         ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full","Half","Empty");  
26         myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());  
27         myComboBox.setItems(options);  
28         Image image=new Image(String.valueOf(JavaFXMLControlsApplication.class.getResource( name: "uon.png")));  
29         myImageView.setImage(image);  
30     }
```

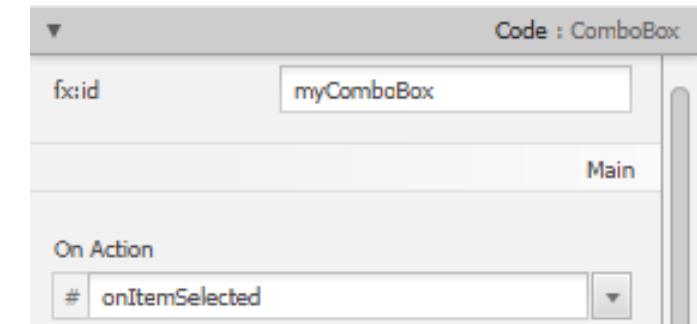
```
3    > import ...  
12  
13    public class JavaFXMLControlsController {  
14  
15  
16    </> @FXML private ComboBox<String> myComboBox;  
17    </> @FXML private ImageView myImageView;  
18    </> @FXML private Label myLabel;  
19    </> @FXML private CheckBox myNoCheckBox;  
20    </> @FXML private CheckBox myYesCheckBox;  
21  
22    @FXML  
23    private void initialize(){  
24        myYesCheckBox.setSelected(true);  
25        ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full","Half","Empty");  
26        myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());  
27        myComboBox.setItems(options);  
28        Image image=new Image(String.valueOf(JavaFXMLControlsApplication.class.getResource( name: "uon.png")));  
29        myImageView.setImage(image);  
30    }  
}
```

ObservableList: A list that allows listeners to track changes when they occur

ComboBox

</>

- How do you get selections?
 - Using SceneBuilder and @FXML
 - Choose OnAction event in SceneBuilder
 - Then write a handler method



```
50      @FXML  
51          private void onItemSelected() {  
52              String s=myComboBox.getSelectionModel().getSelectedItem();  
53              System.out.println("Choice: "+s);  
54      }
```

ComboBox

</>

- There is another way to get the choice from a combo box, and in fact to link other kinds of controls with variables: Binding

```
ObservableList<String> options= FXCollections.observableArrayList( ...es: "Full", "Half", "Empty");
myLabel.textProperty().bind(myComboBox.getSelectionModel().selectedItemProperty());
myComboBox.setItems(options);
```

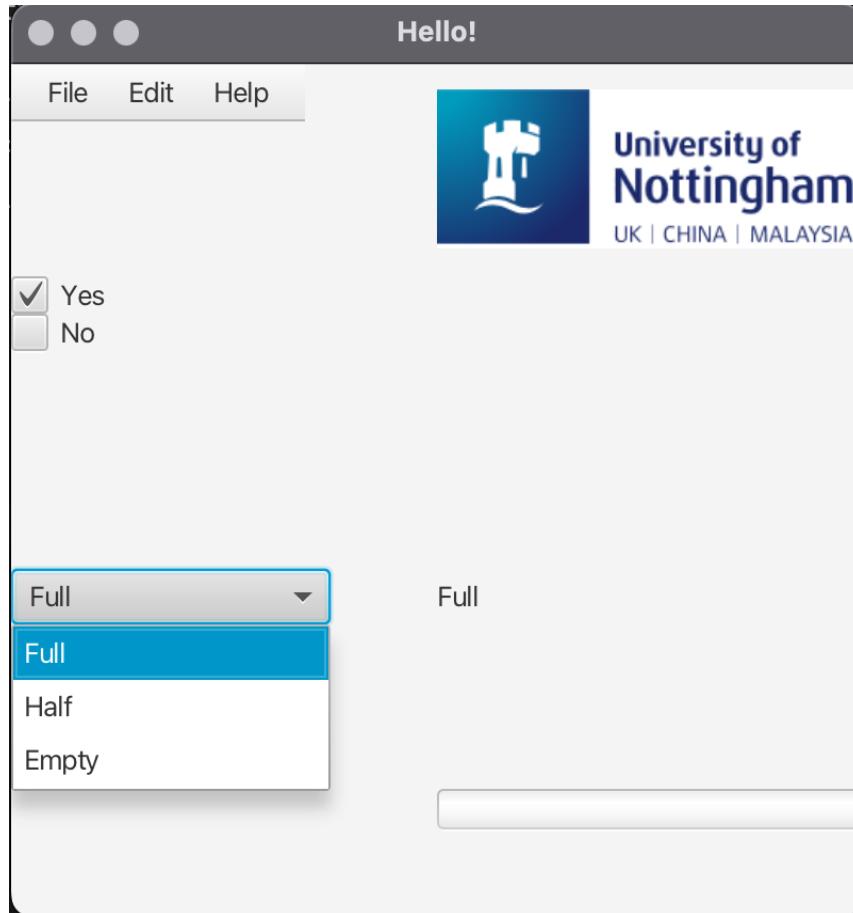
- Whenever the value of the ComboBox property changes, the Label text changes automatically



ComboBox

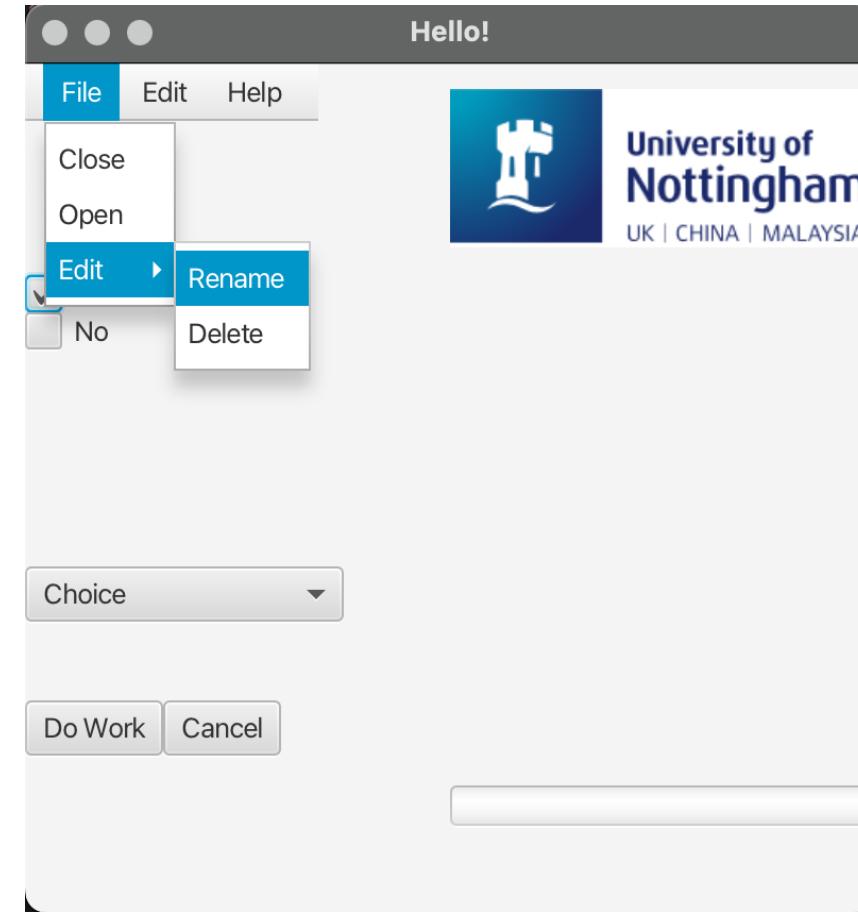
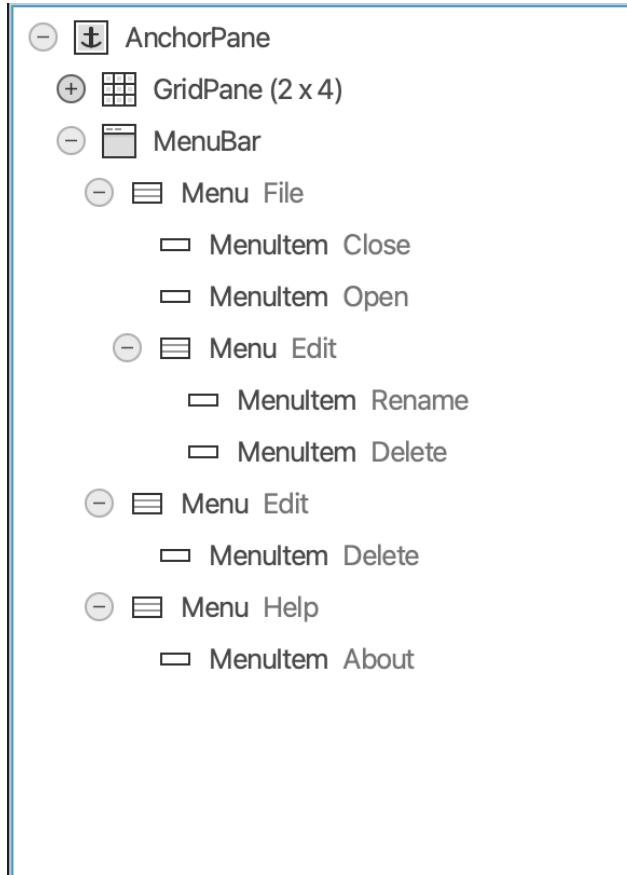
</>

- The result



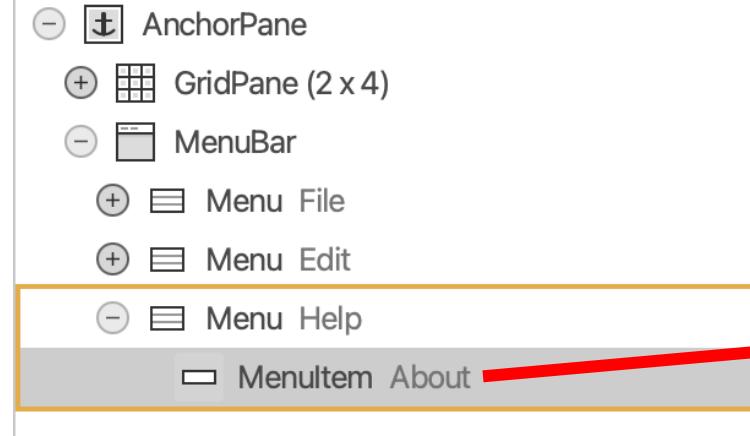
MenuBar

</>



MenuBar

</>



The properties panel on the right shows the following settings for the selected MenuItemAbout node:

- Identity: fx:id = myMenuItemAbout
- Main
- On Action: # onAboutClicked

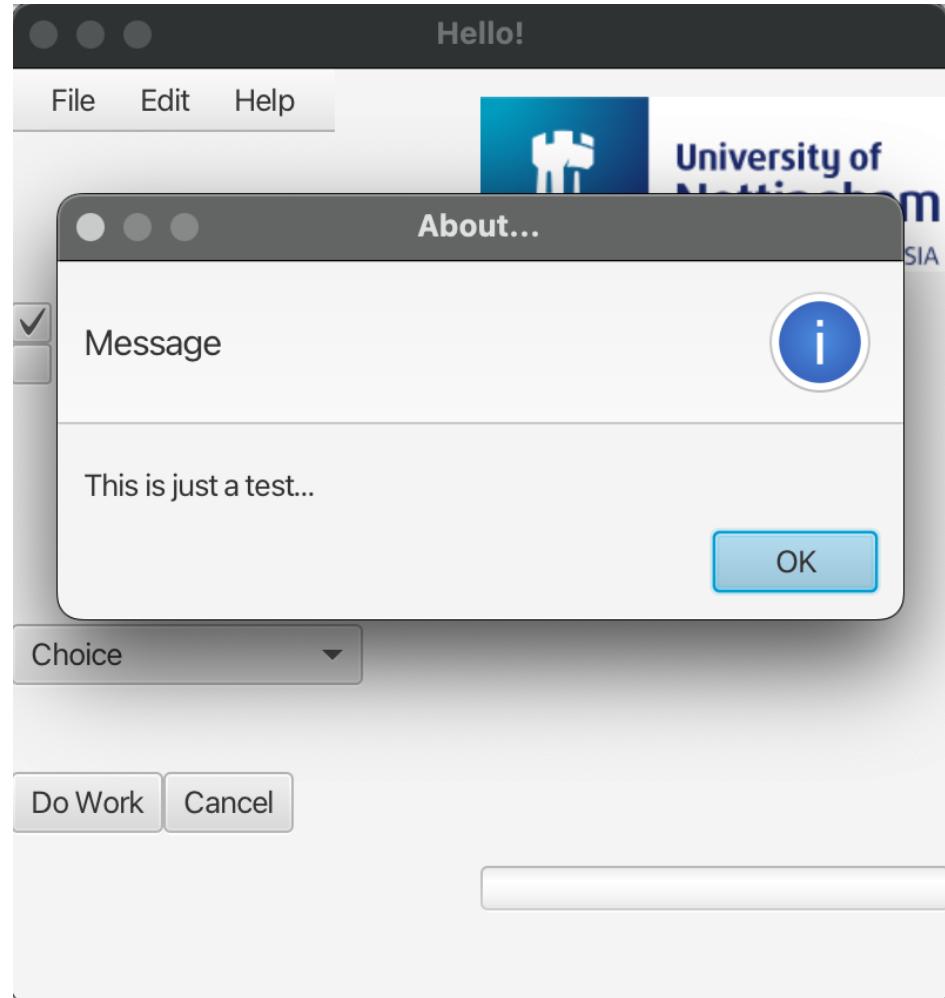
A red arrow points from the 'On Action' field to the corresponding code in the editor.

```
55 @FXML
56 private void onAboutClicked() {
57     Alert alert=new Alert(Alert.AlertType.INFORMATION);
58     alert.setTitle("About...");
59     alert.setContentText("This is just a test...");
60     alert.show();
61 }
```

MenuBar

</>

- The result



Alert Types

</>

```
55     @FXML  
56     private void onAboutClicked() {  
57         Alert alert=new Alert(Alert.AlertType.INFORMATION);  
58         Alert alert2 = new Alert(Alert.AlertType.)  
59         alert.setTitle("About...");  
60         alert.setContentText("This is just a t  
61         alert.show();  
62     }  
63 }  
64 }
```

(f) INFORMATION
valueOf(String name)
(f) CONFIRMATION
(f) ERROR
(f) NONE
(f) WARNING

AlertType
AlertType
AlertType
AlertType
AlertType
AlertType

FXML

Styling

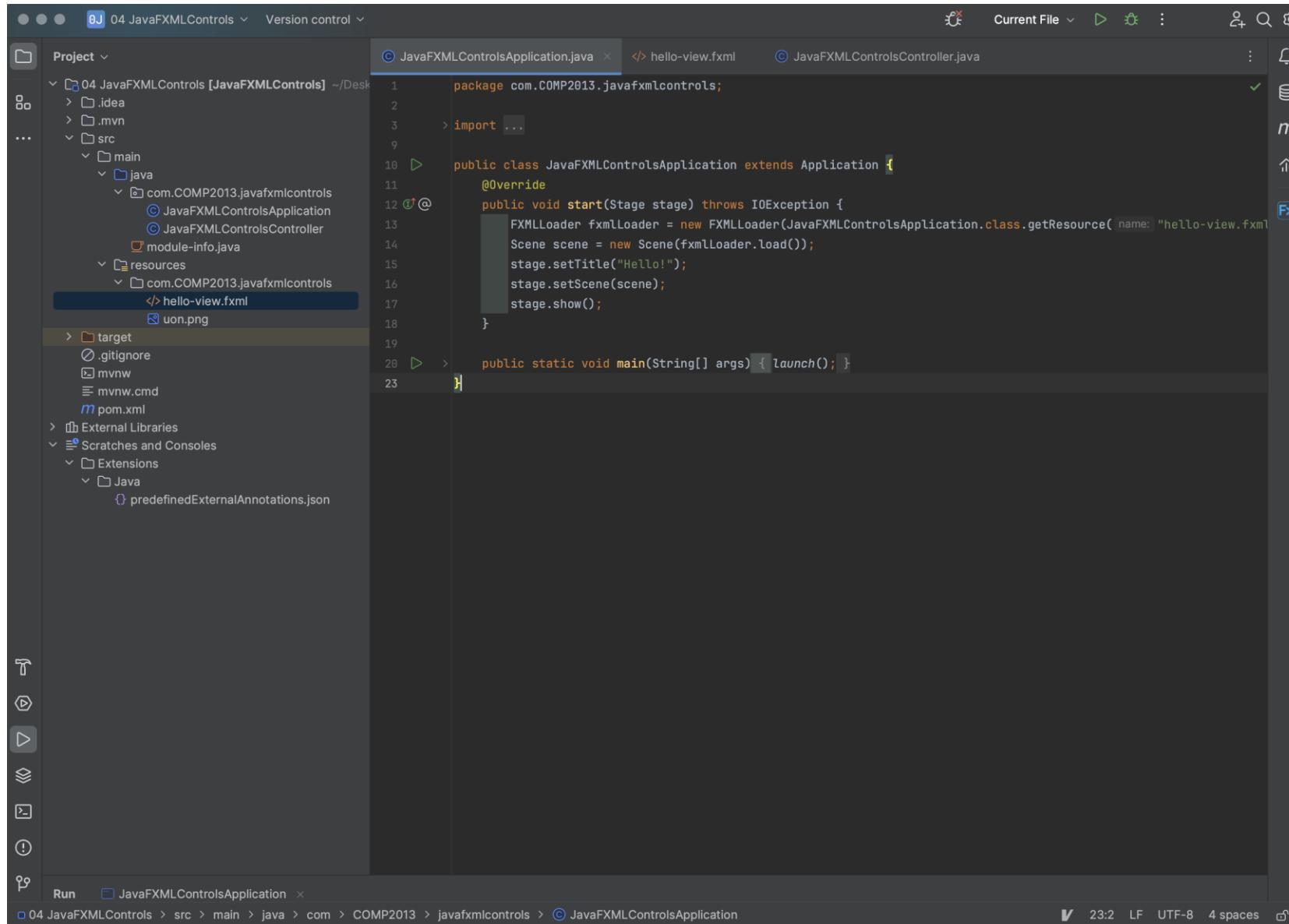
Customising GUI Appearance

</>

- In addition to laying out the elements on the screen, we can also specify how they look
- Remember we want to keep appearance, design and code separated
- Here we will change the appearance without adjusting anything else.
 - Suppose your company acquires a software package
 - The first thing you might want to do is bring the appearance in line with the rest of your product range

Applications with Distinctive Themes

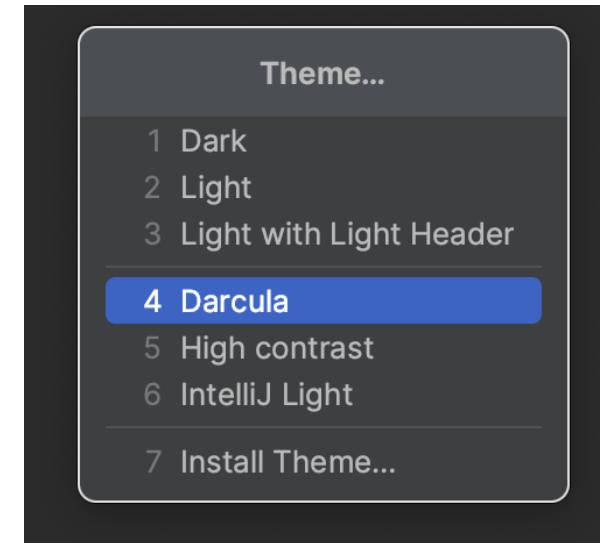
</>



The screenshot shows an IDE interface with a dark theme. On the left is the Project tool window displaying a file structure for a JavaFX project named "04 JavaFXMLControls". The main editor window contains the following Java code:

```
package com.COMP2013.javafxcontrols;
import ...
public class JavaFXMLControlsApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(JavaFXMLControlsApplication.class.getResource( name: "hello-view.fxml" ));
        Scene scene = new Scene(fxmlLoader.load());
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) { launch(); }
}
```

The file "hello-view.fxml" is selected in the Project tool window.



- A Cascading Style Sheet (CSS) is a means to define the look (visual properties) of UI elements in a GUI application
 - Primary developed for use in web pages
 - Allows separation of presentation and content/behaviour
- CSS provides the syntax to define rules
 - A rule consists of **selector** and **property/value** pair(s)

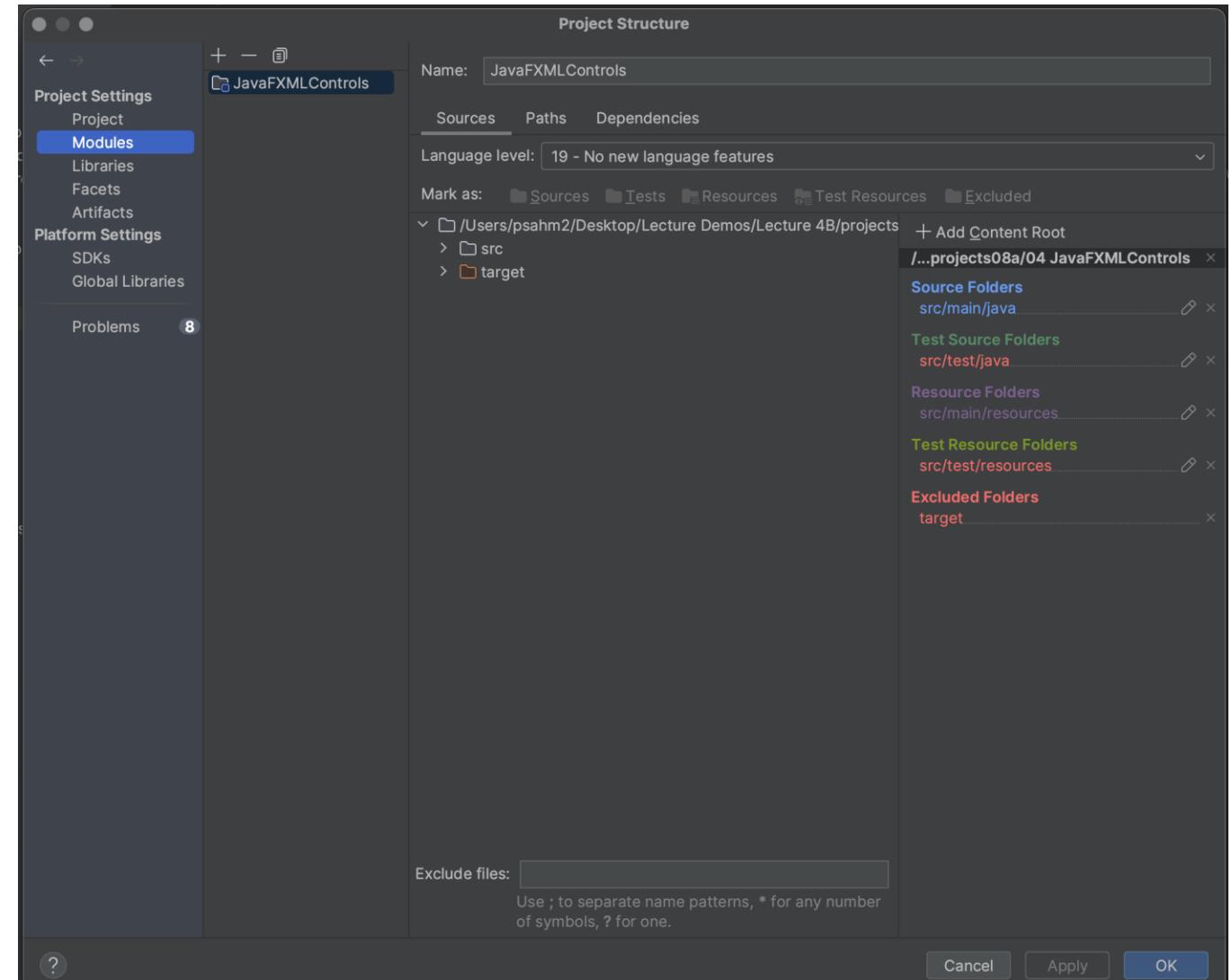
```
.button {  
    -fx-background-color: red;  
    -fx-text-size: 22;  
}
```

<https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html>

User Defined Style Sheets

</>

- Create *.css style sheet in "resources" folder
 - Remember to add the resources folder to the "build-path"



User Defined Style Sheets

</>

- JavaFX CSS naming conventions
 - All style class names are lowercase node names
 - If node name consists of multiple words style class names use hyphen
 - Property names start with "-fx-" followed by instance variables (lowercase + hyphenated)

```
.check-box .box{  
    -fx-background-color:white;  
    -fx-border-color:red;  
    -fx-border-radius:12px;  
}  
.check-box{  
    -fx-font-size:16;  
}
```

User Defined Style Sheets

</>

```
30 >     public class JavaFXMLCSSApplication extends Application {  
31  
32         @Override  
33 @  
34     public void start(Stage primaryStage) throws Exception{  
35         Parent root = FXMLLoader.load(getClass().getResource( name: "javafxmcss-view.fxml"));  
36         root.setStyle("-fx-font-size:20");  
37         primaryStage.setTitle("Hello JavaFXMLCSS");  
38         Application.setUserAgentStylesheet(Application.STYLESHEET_CASPIAN);  
39         Scene scene=new Scene(root);  
40         scene.getStylesheets().add(String.valueOf(getClass().getResource( name: "mystyles.css")));  
41         primaryStage.setScene(scene);  
42         primaryStage.show();  
43     }  
44  
45 >     public static void main(String[] args) { launch(args); }  
46 }  
47 }
```

User Defined Style Sheets

</>

```
30 >     public class JavaFXMLCSSApplication extends Application {  
31  
32         @Override  
33 @  
34     public void start(Stage primaryStage) throws Exception{  
35         Parent root = FXMLLoader.load(getClass().getResource( name: "javafxmcss-view.fxml"));  
36         root.setStyle("-fx-font-size:20");  
37         primaryStage.setTitle("Hello JavaFXMLCSS");  
38         Application.setUserAgentStylesheet(Application.STYLESHEET_CASPIAN);  
39         Scene scene=new Scene(root);  
40         scene.getStylesheets().add(String.valueOf(getClass().getResource( name: "mystyles.css")));  
41         primaryStage.setScene(scene);  
42         primaryStage.show();  
43     }  
44  
45 >     public static void main(String[] args) { launch(args); }  
46 }  
47 }
```

User Defined Style Sheets

</>

```
30 >     public class JavaFXMLCSSApplication extends Application {  
31  
32         @Override  
33 @  
34     public void start(Stage primaryStage) throws Exception{  
35         Parent root = FXMLLoader.load(getClass().getResource( name: "javafxmcss-view.fxml"));  
36         root.setStyle("-fx-font-size:20");  
37         primaryStage.setTitle("Hello JavaFXMLCSS");  
38         Application.setUserAgentStylesheet(Application.STYLESHEET_CASPIAN);  
39         Scene scene=new Scene(root);  
40         scene.getStylesheets().add(String.valueOf(getClass().getResource( name: "mystyles.css")));  
41         primaryStage.setScene(scene);  
42         primaryStage.show();  
43     }  
44  
45 >     public static void main(String[] args) { launch(args); }  
46 }  
47 }
```

User Defined Style Sheets

</>

```
30 >     public class JavaFXMLCSSApplication extends Application {  
31  
32         @Override  
33 @  
34     public void start(Stage primaryStage) throws Exception{  
35         Parent root = FXMLLoader.load(getClass().getResource( name: "javafxmcss-view.fxml"));  
36         root.setStyle("-fx-font-size:20");  
37         primaryStage.setTitle("Hello JavaFXMLCSS");  
38         Application.setUserAgentStylesheet(Application.STYLESHEET_CASPIAN);  
39         Scene scene=new Scene(root);  
40         scene.getStylesheets().add(String.valueOf(getClass().getResource( name: "mystyles.css")));  
41         primaryStage.setScene(scene);  
42         primaryStage.show();  
43     }  
44  
45 >     public static void main(String[] args) { launch(args); }  
46 }  
47 }
```

User Defined Style Sheets

```
30 >     public class JavaFXMLCSSApplication extends Application {  
31  
32  
33 @ 34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45 >     public static void main(String[] args) { launch(args); }  
46  
47 }
```

The screenshot shows a Java application window titled "Hello JavaFXMLCSS" running in a Java IDE. The code in the editor is as follows:

```
public class JavaFXMLCSSApplication extends Application {  
    public static void main(String[] args) { launch(args); }  
}
```

The application's user interface consists of two windows. Each window has a title bar "Hello JavaFXMLCSS" and a menu bar with "File", "Edit", and "Help". Inside each window, there is a "Label", a "dropdown menu", and two "Buttons". A red box highlights the "Label" and "dropdown menu" components. A yellow box highlights the "Buttons". A red circle with a checkmark surrounds the "Yes" radio button in both windows, indicating it is selected.

User Defined Style Sheets

</>

```
30 >     public class JavaFXMLCSSApplication extends Application {  
31  
32         @Override  
33 @  
34     public void start(Stage primaryStage) throws Exception{  
35         Parent root = FXMLLoader.load(getClass().getResource( name: "javafxmcss-view.fxml"));  
36         root.setStyle("-fx-font-size:20");  
37         primaryStage.setTitle("Hello JavaFXMLCSS");  
38         Application.setUserAgentStylesheet(Application.STYLESHEET_CASPIAN);  
39         Scene scene=new Scene(root);  
40         scene.getStylesheets().add(String.valueOf(getClass().getResource( name: "mystyles.css")));  
41         primaryStage.setScene(scene);  
42         primaryStage.show();  
43  
44  
45 >     public static void main(String[] args) { launch(args); }  
46  
47 }
```

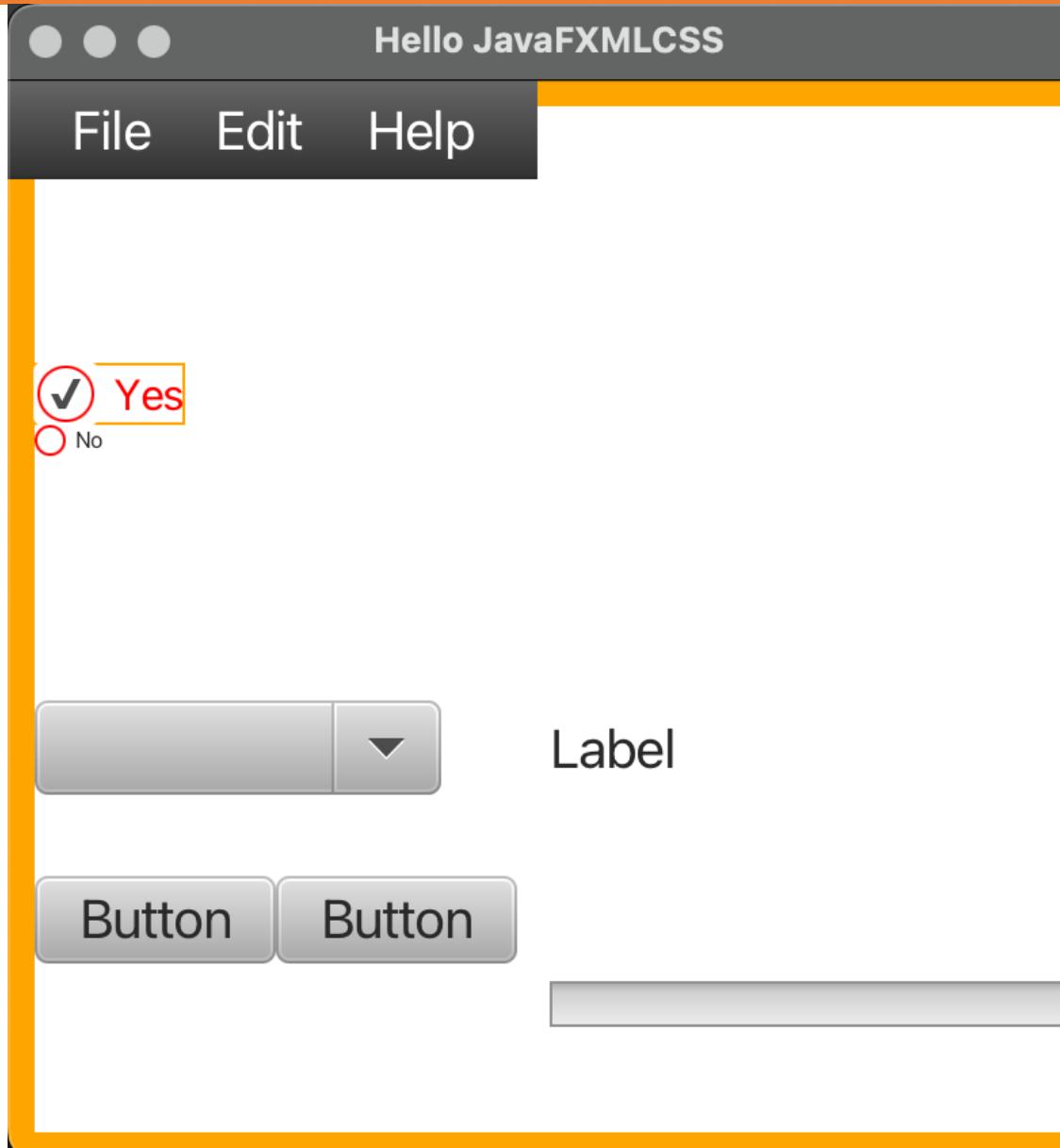
Mystyles.css file

</>

```
1   .check-box{  
2     -fx-background-color:white;  
3     -fx-border-color:red;  
4     -fx-border-radius:12px;  
5   }  
6   .check-box{  
7     -fx-font-size:16;  
8   }  
9   #myNoCheckBox{  
10    -fx-font-size:8;  
11  }  
12  |
```

User Defined Style Sheets

- The result



Default Style Sheets

</>

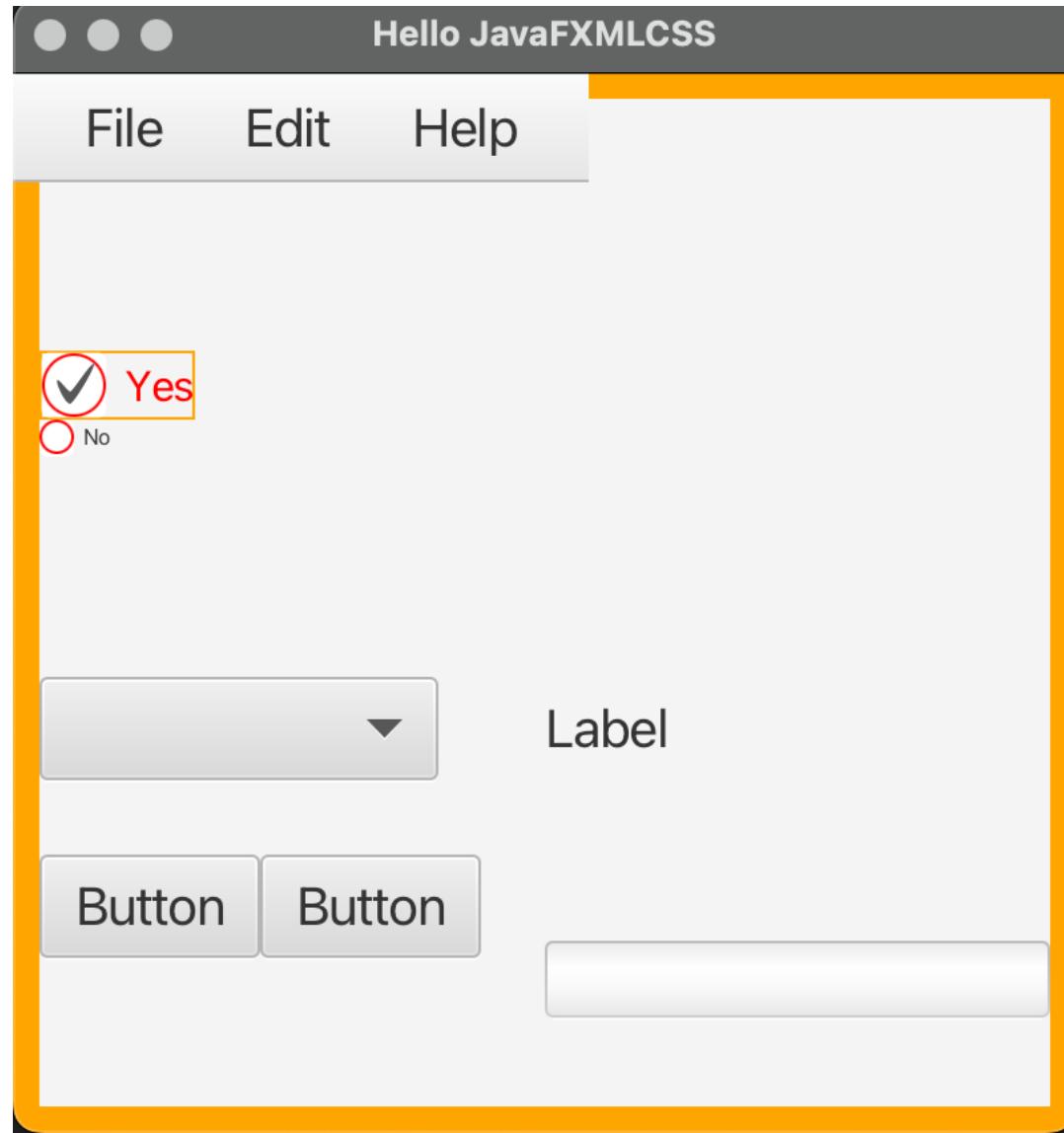
- Default look that you get for JavaFX application is "modena.css"

```
Application.setUserAgentStylesheet(Application.STYLESHEET_CASPIAN) ;  
primaryStage.setScene(scene) ;  
primaryStage.show() ;
```

Default Style Sheets

</>

- The result



Adding Inline Styles

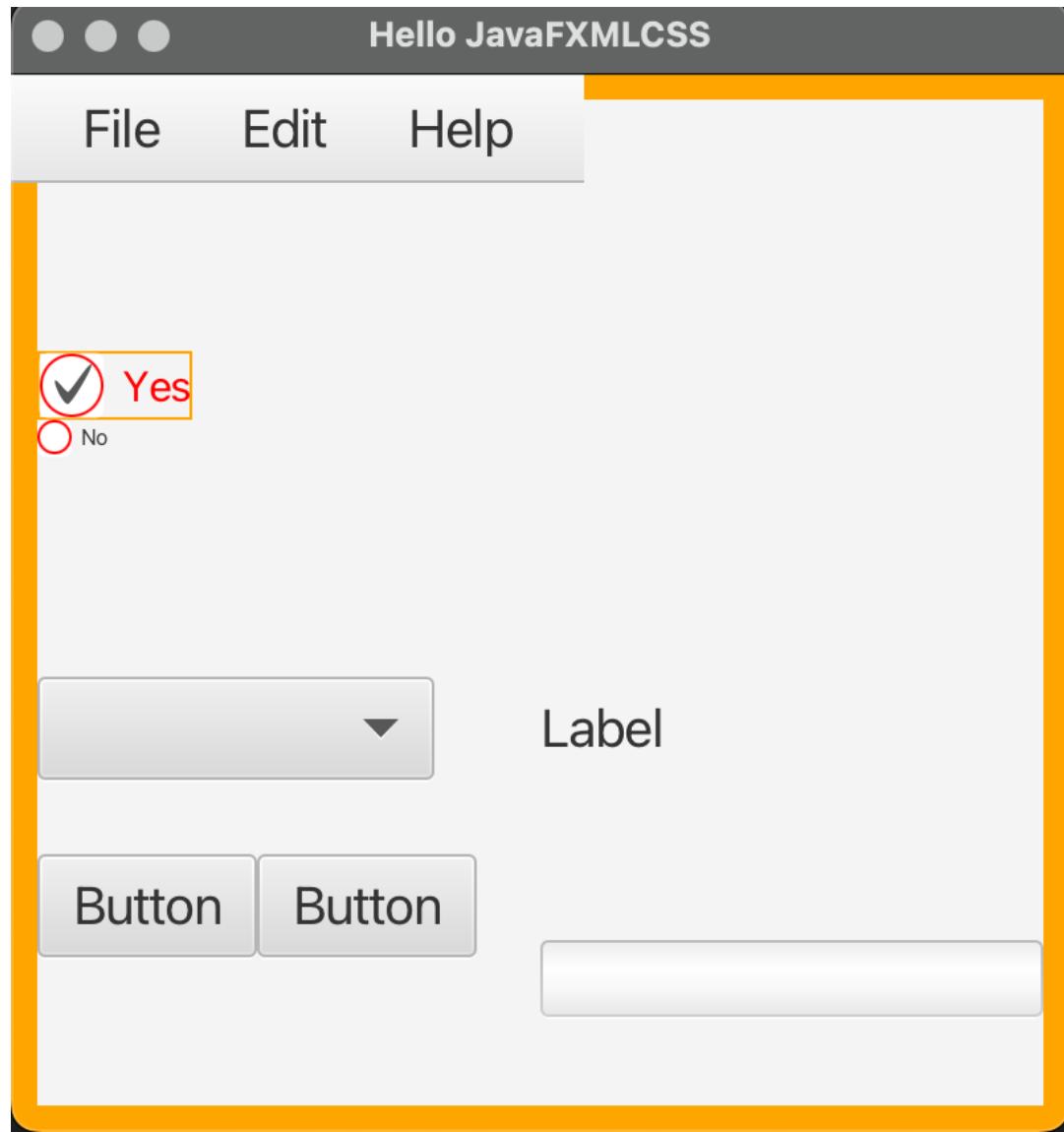
</>

- You can add styles to individual nodes
 - Use: `setStyle(String inlineStyle)` to set the style and `getStyle()` to get the style of a node

```
public class Controller {  
    @FXML private CheckBox myYesCheckBox;  
  
    @FXML  
    private void initialize(){  
        myYesCheckBox.setSelected(true);  
        myYesCheckBox.setStyle("-fx-text-fill:red;-fx-font-weight:bold");  
    }  
}
```

Adding Inline Styles

</>



```
        .  
        (myGridPane.getParent());  
        "-fx-border-width:10;-fx-border-color:orange");  
        selected(true);  
        Le("-fx-text-fill:red;-fx-font-weight:bold;-fx-border-color:inherit");
```

Priorities of Styles

</>

- Ranking order (highest to lowest)
 - Inline style
 - Parent style sheet
 - Scene style sheet
 - Values set in the code using JavaFX API (e.g. using "setFont()")
 - User agent style sheet (JavaFX runtime)

Using ID selectors

</>

- We can also use the IDs created in Scene Builder to set styles for individual nodes in the CSS style sheet

```
.check-box{  
-fx-font-size:16;  
}
```

```
#myNoCheckBox{  
-fx-font-size:8;  
}
```



</>



Properties : CheckBox

Text

Text	No
Font	System 13px
Text Fill	BLACK
Wrap Text	<input type="checkbox"/>
Text Alignment	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Text Overrun	ELLIPSIS
Ellipsis String	...
Underline	<input type="checkbox"/>
Line Spacing	0

Specific

Allow Indeterminate	<input type="checkbox"/>
Selected	<input type="checkbox"/>
Indeterminate	<input type="checkbox"/>

Graphic

Graphic Text Gap	4
------------------	---

Layout : CheckBox

Code : CheckBox

File Edit Help

Yes

No

Label

Button Button

COMP2013

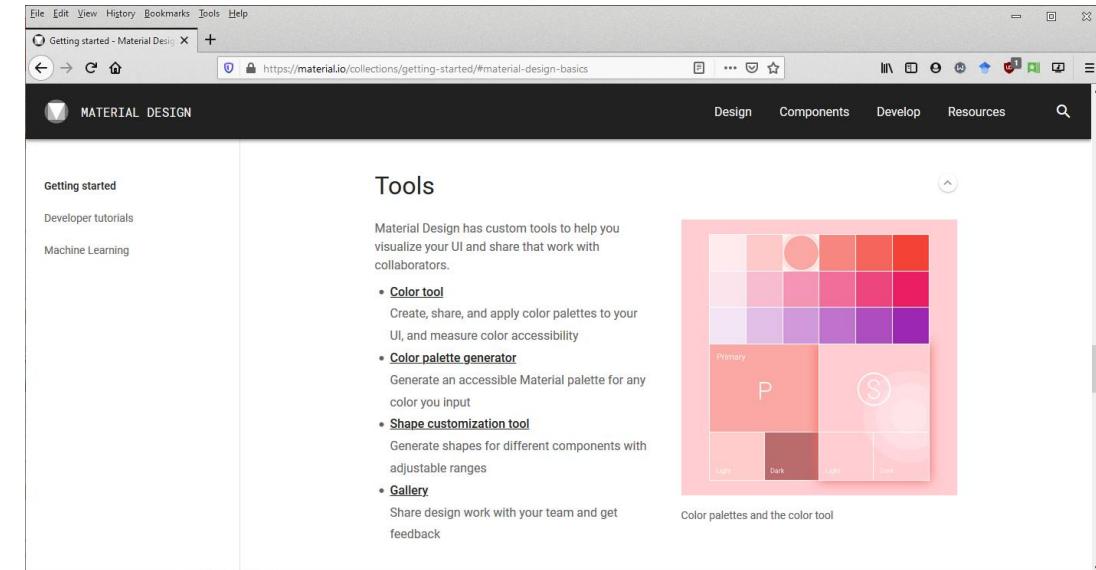
The interface shows a window with a menu bar (File, Edit, Help) and a central panel containing a checkbox group ('Yes' and 'No') with a yellow border, a dropdown menu, a label, and two buttons. To the right is a properties panel for a 'CheckBox' component, divided into sections for Text, Specific, and Graphic. The 'Text' section includes fields for Text, Font, Text Fill, Wrap Text, Text Alignment, Text Overrun, Ellipsis String, Underline, and Line Spacing. The 'Specific' section includes checkboxes for Allow Indeterminate, Selected, and Indeterminate. The 'Graphic' section includes a Graphic Text Gap field set to 4. Below the properties panel are 'Layout : CheckBox' and 'Code : CheckBox' buttons. A red circle highlights the properties panel, and a blue speech bubble with '</>' is in the top right corner.

Resources

</>

- General JavaFX
 - Comprehensive introduction to JavaFX
 - https://www3.ntu.edu.sg/home/ehchua/programming/java/javafx1_intro.html

- Excellent material design guide
 - <https://material.io/design/>



some final remarks ...



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



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/>



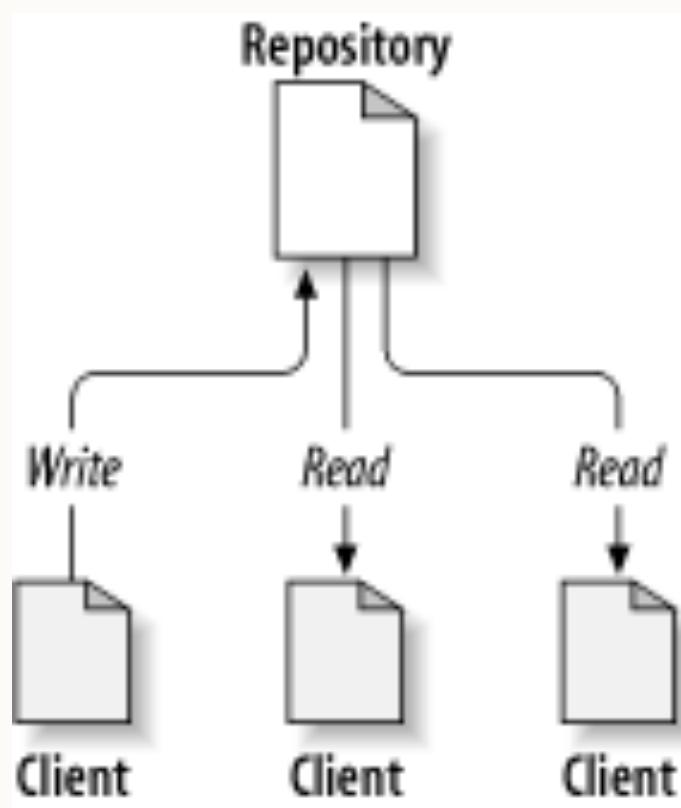
Why Use Version Control?

- . 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



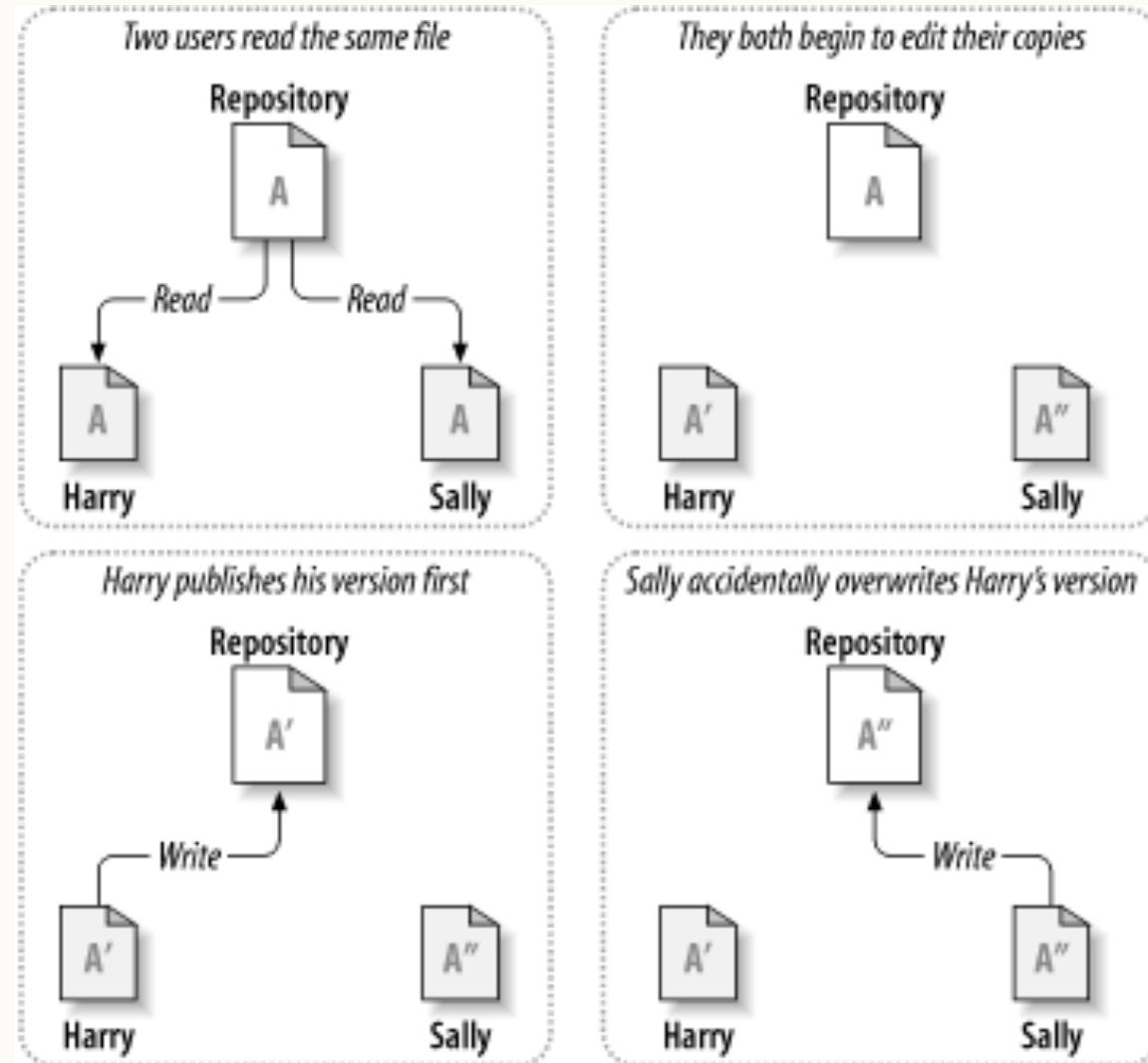


Concurrency Management

- 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

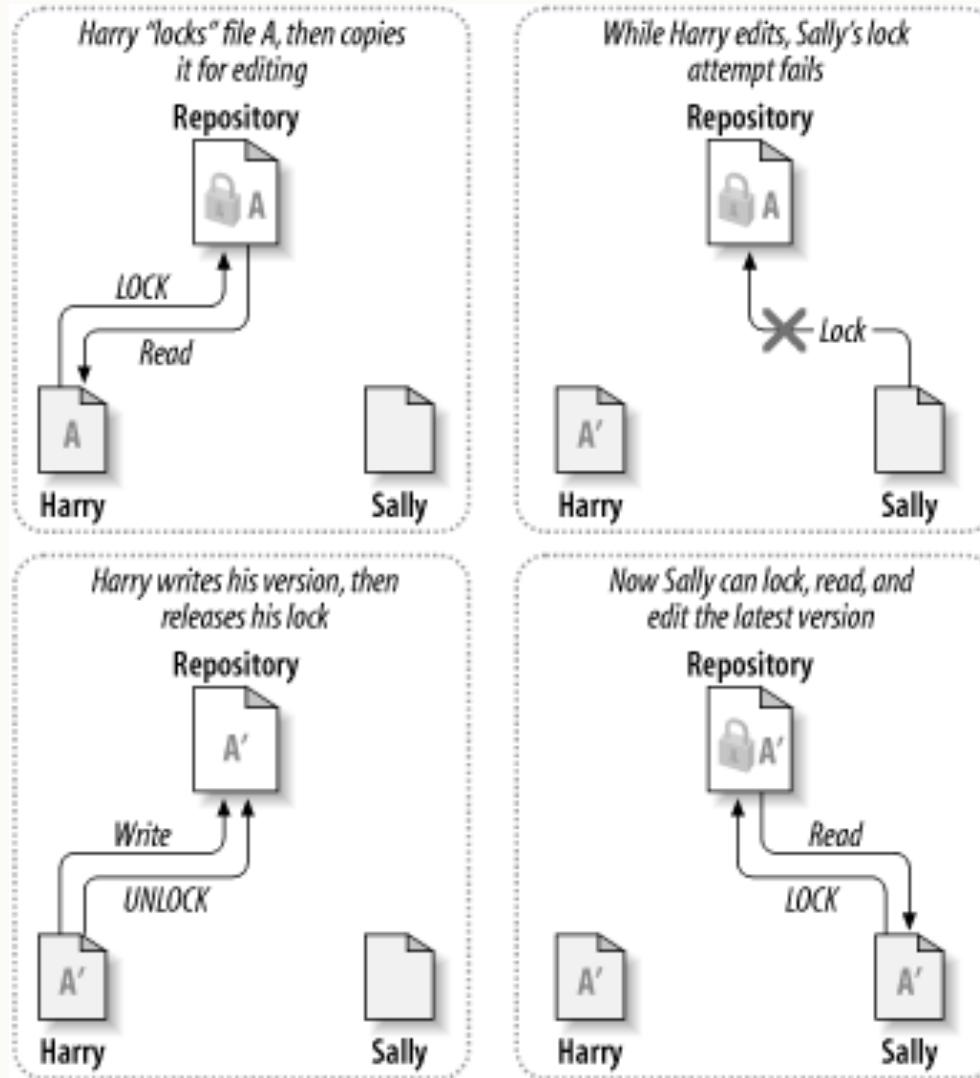


The Problem of File Sharing





The Lock-Modify-Unlock Solution





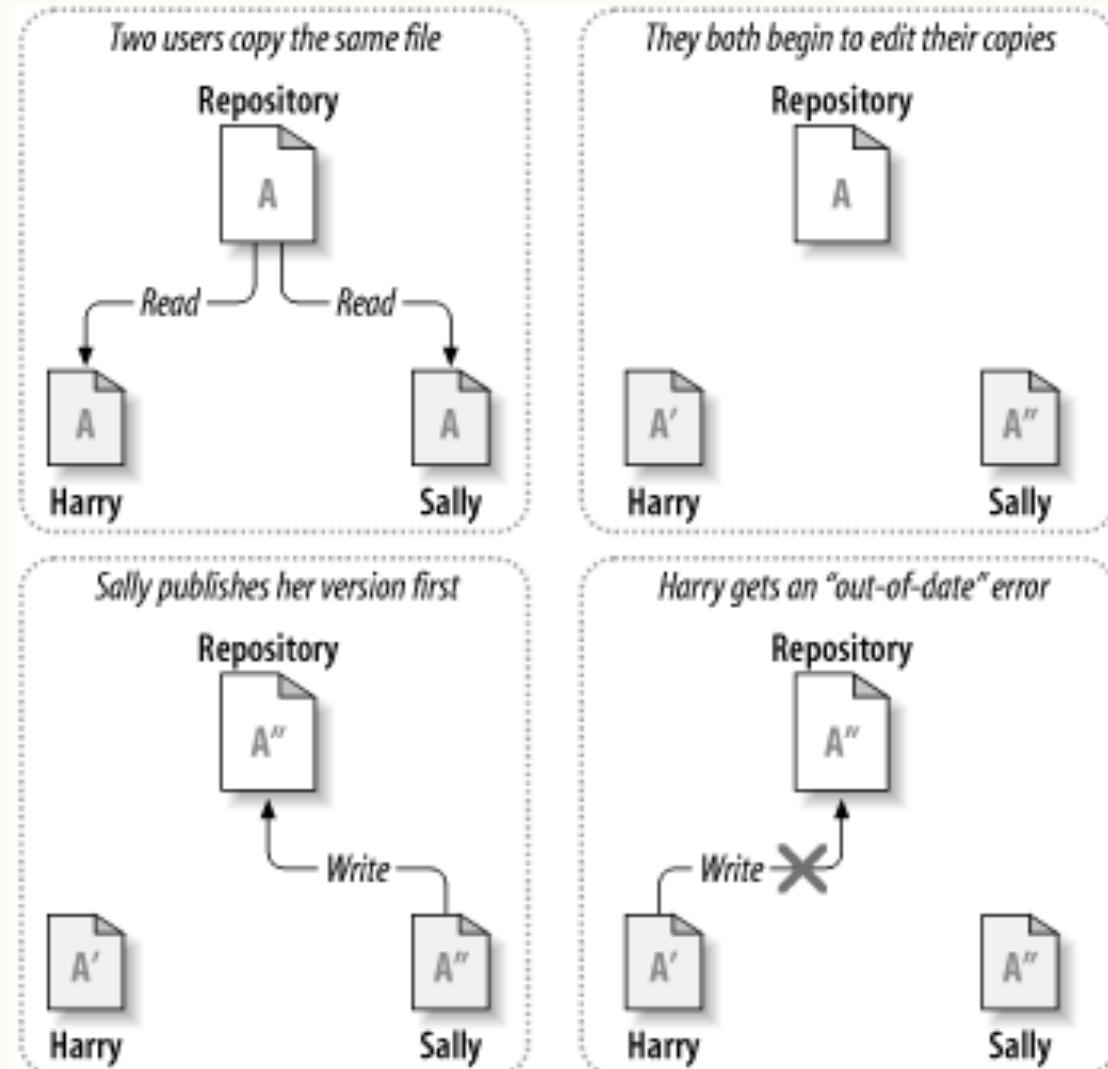
The Lock-Modify-Unlock Solution

- 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.

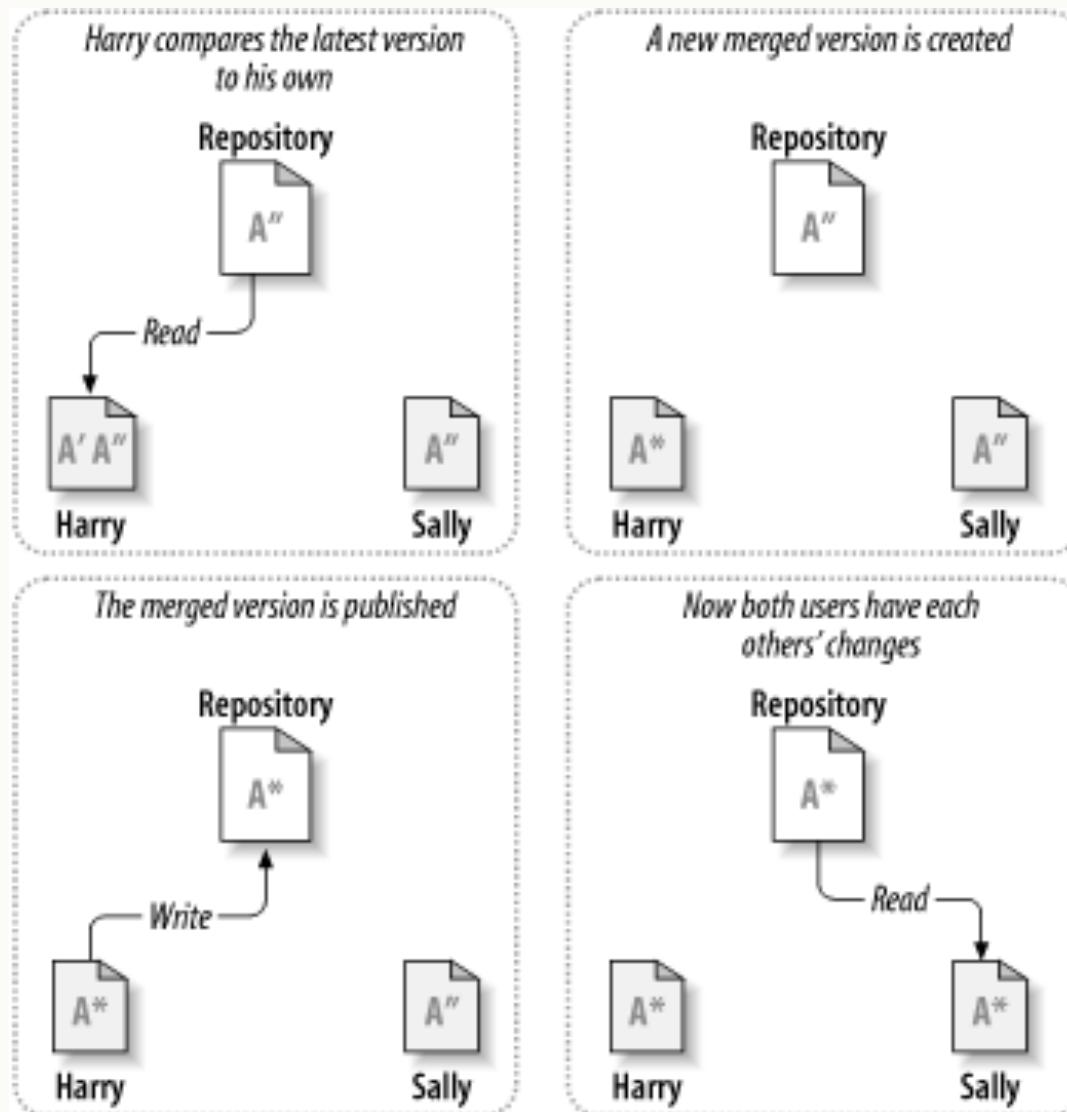


The Copy-Modify-Merge Solution





The Copy-Modify-Merge Solution





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

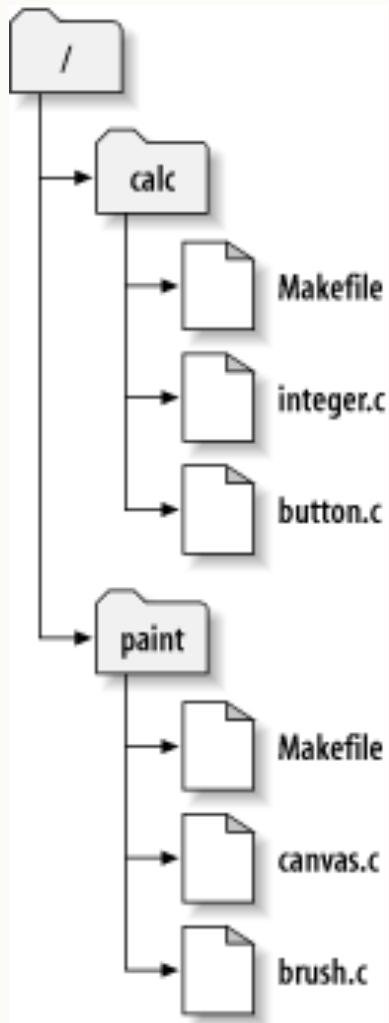


Working Copies

- . 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
```

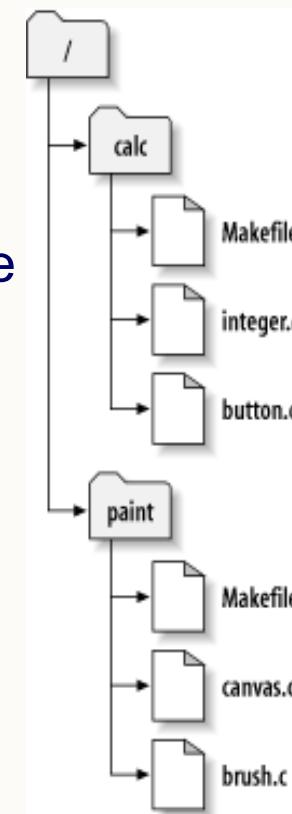
```
Checked out revision 56.
```

```
$ 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



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 many 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;
```



Rule 10: Example

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

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

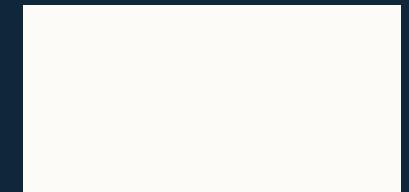
    if (debug) {
        cerr << "RSL_OglTexture::CopyImageData() name: " << this->GetName() << endl; cerr
        << " width: " << this->GetWidth() << ", height: " << this->GetHeight() << endl;
    }
    for (currentRow = 0; currentRow < heightOfOneColumn; currentRow++) { for
        (currentCol = 0; currentCol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



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>

The screenshot shows a Java code editor window titled "Painter.java". The code is a simple Java Swing application. The Javadoc comments at the top of the file provide details about the file, author, date, and source. The code uses Java's Border Layout and JFrame to create a paint panel and a label. It also sets the frame size and makes it visible.

```
/*
 * @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.JPanel;

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 */
        JPanel paintPanel = new JPanel();
        /** 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)



Lecture 06B

Maintainable GUI Development (2/2)

JavaFX advanced topics: Multi-threading and animation



Horia A. Maior and Marjahan Begum

</>



Lecture 06B

More Maintainable GUI Development (2/2)

JavaFX advanced topics: Multi-threading and animation

Horia A. Maior and Marjahan Begum

Topics for this Week

</>

- Lecture 05A
 - Build Tools
 - JavaFX(ML) advanced topics: controls and styling
- Lecture 05B
 - CW release
- Lecture 06A
 - Setting up Git for your project
- Lecture 06B
 - JavaFX advanced topics: Multi-threading and animation

Request Access to UoN Git before Friday (Tomorrow)

</>



<https://forms.office.com/e/ieDqqj1JUu>

JavaFX advanced topics

Multi-threading: Dealing with unresponsive GUIs

Stopping GUIs becoming unresponsive

</>

- The facts:
 - JavaFX launches the UI on a JavaFX Application thread
 - This thread should be left handling the UI interaction
 - Heavy computation should be done elsewhere to prevent freezing!
- JavaFX provides a solution:
 - Package "javafx.concurrent"
 - A way for JavaFX to create and communicate with other threads

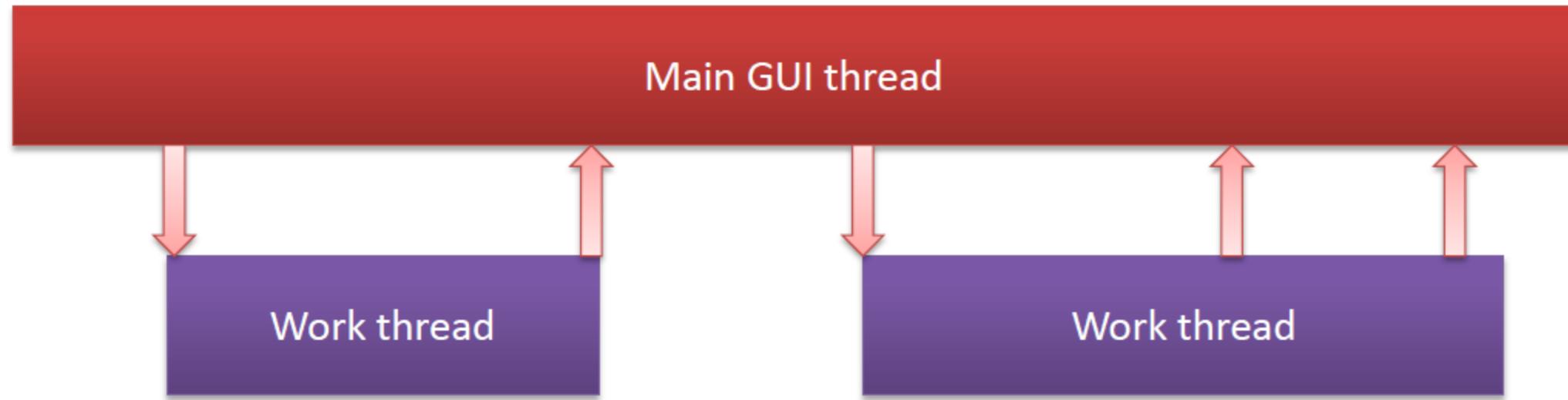
JavaFX Application Thread

</>

```
© JavaFXMLControlsApplication.java × m pom.xml (JavaFXMLControlsMultiThreads) © JavaFXMLControlsController.java © Pro...  
1 package com.comp2013.javafxcontrolsmultithreads; ✓ 1  
2  
3 > import ...  
4  
5  
10 ▶ public class JavaFXMLControlsApplication extends Application {  
11     @Override  
12     public void start(Stage stage) throws IOException {  
13         FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(name: "javafxcontrols-view.fxml"));  
14         Scene scene = new Scene(fxmlLoader.load());  
15         stage.setTitle("Hello Java FXML Controls!");  
16         stage.setScene(scene);  
17         stage.show();  
18     }  
19  
20 ▶     public static void main(String[] args) {  
21         launch();  
22     }  
23 }
```

Stopping GUIs becoming unresponsive

</>

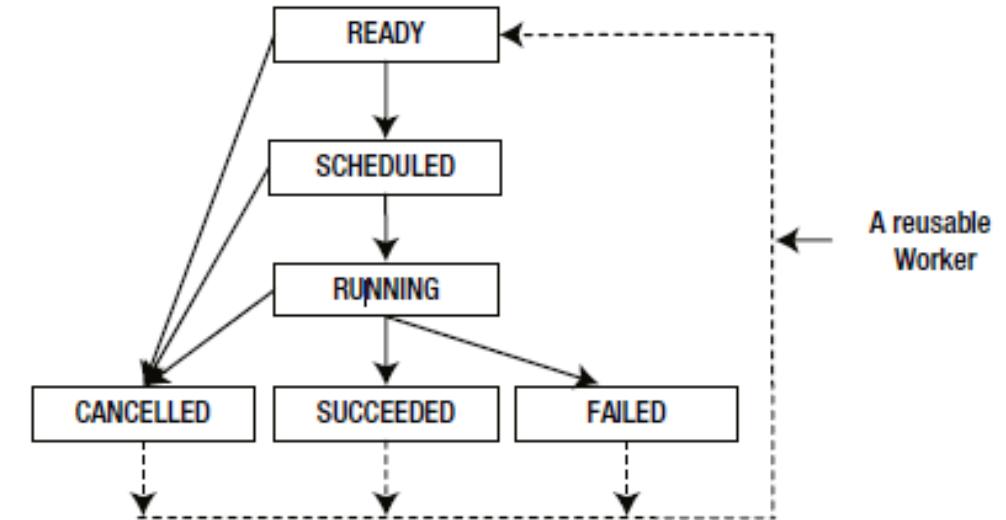
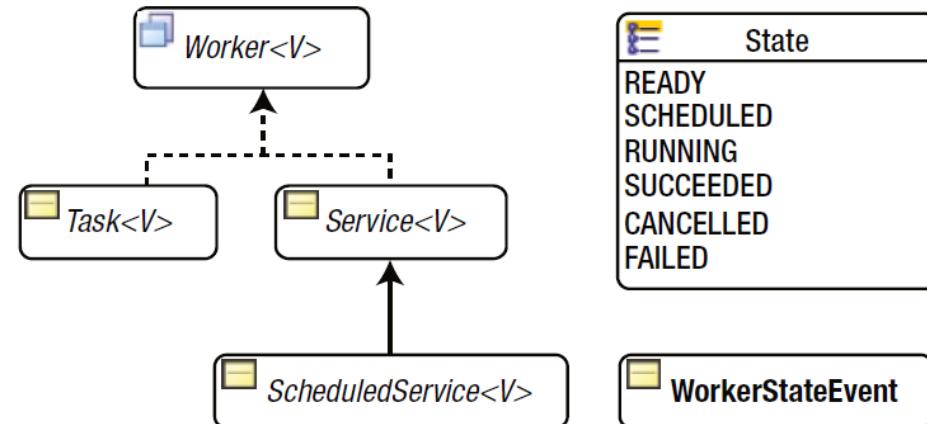


- JavaFX provides some helper classes

Concurrency Framework in Java

</>

- Concurrency
 - The ability of different parts of a program to be executed out-of-order or in partial order, without affecting the final outcome
 - `java.util.concurrent` package
- Classes in the JavaFX Concurrency Framework



Source: Learn JavaFX 8 - Building User Experience and Interfaces with Java 8 (Apress.2015)

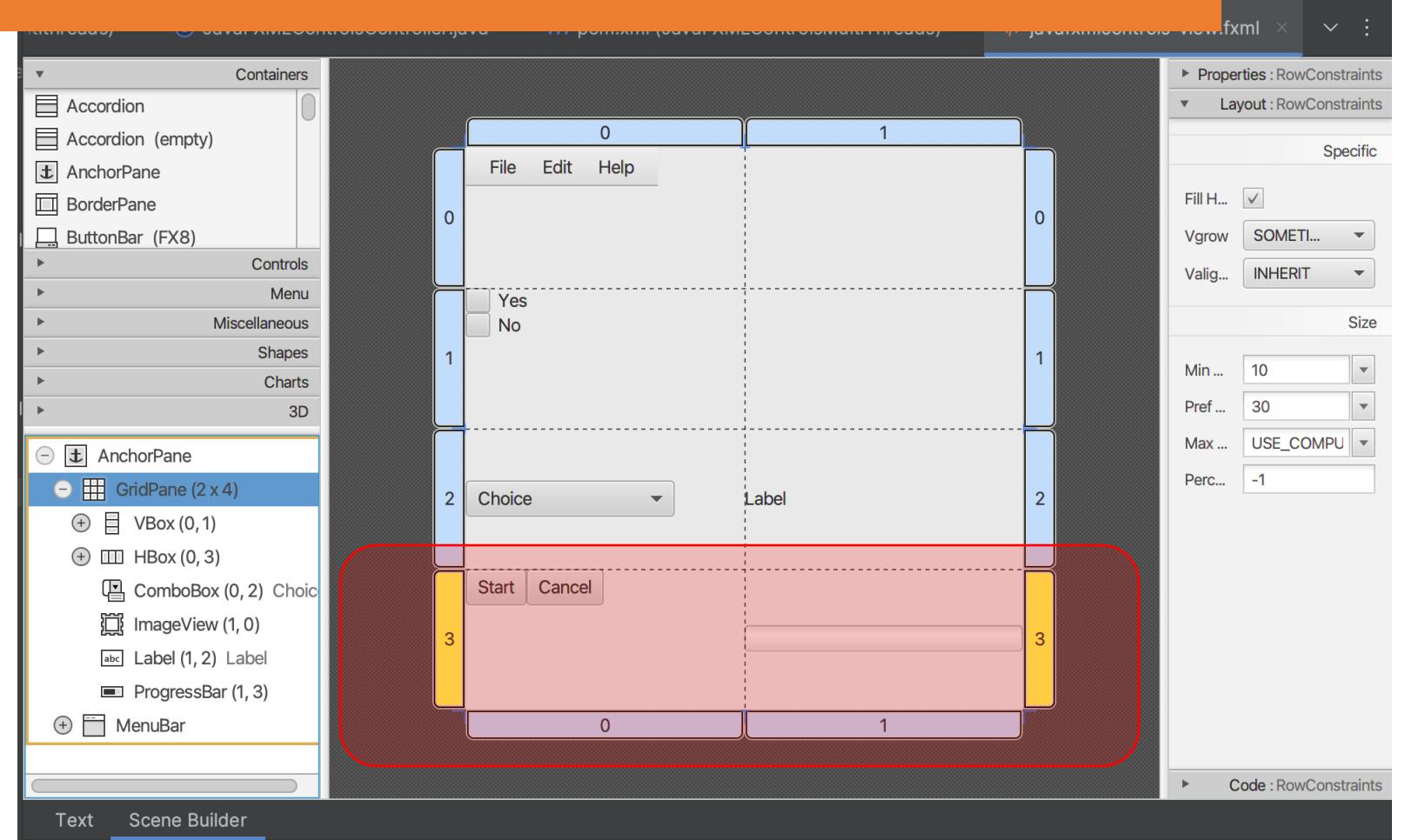
Concurrency Framework in JavaFX

</>

- Framework consist of one interface + four classes + one enum
 - **Worker** interface specifies the methods available to background (work) threads
 - **Task** class instance represents a one-shot task
 - **Service** class instance represents a reusable task
 - **ScheduledService** class instance represents a reusable task that runs repeatedly following a specified interval
 - **WorkerStateEvent** class instance represents an event that occurs as state of Worker changes
 - You can add event handlers to all three types of tasks to listen to the changes in their states
 - **State** enum constants represents different states of a worker

"Task

</>



Let's build a progress bar to show some work done in another thread

"Task" Example

</>

- Worker interface
 - Specifies the methods available to **background threads** when working with JavaFX
 - Various useful methods such as isRunning(), getProgress(), cancel()...
 - <https://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/Worker.html>
- Task class
 - Abstract class which implements the Worker interface
 - We program the actual work that needs to be done on a separate thread
 1. Extend the Task class
 2. Implement call() to do the work
 - Don't directly touch UI components from here
 - Can update the UI with updateProgress(), updateMessage(), updateTitle() methods
 3. Start a new thread, passing the relevant "Worker" as a parameter
 - <https://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/Task.html>

</>

The screenshot shows a JavaFX application builder interface. On the left is a preview window displaying a window with a menu bar (File, Edit, Help) and several UI components: a ChoiceBox labeled "Choice", a Label labeled "Label", a Button labeled "Start" with a red border, and a Button labeled "Cancel". A red box highlights the "Start" button. On the right is a properties panel for a selected "Button" component. The properties are organized into sections: Identity, Main, and various event handlers. The "On Action" section is highlighted with a red box and contains the value "# doWork". Other event sections include On Drag Detected, On Drag Done, On Drag Dropped, On Drag Entered, On Drag Exited, and On Drag Over.

Properties : Button

Layout : Button

Code : Button

Identity

fx:id

Main

On Action

doWork

DragDrop

On Drag Detected

On Drag Done

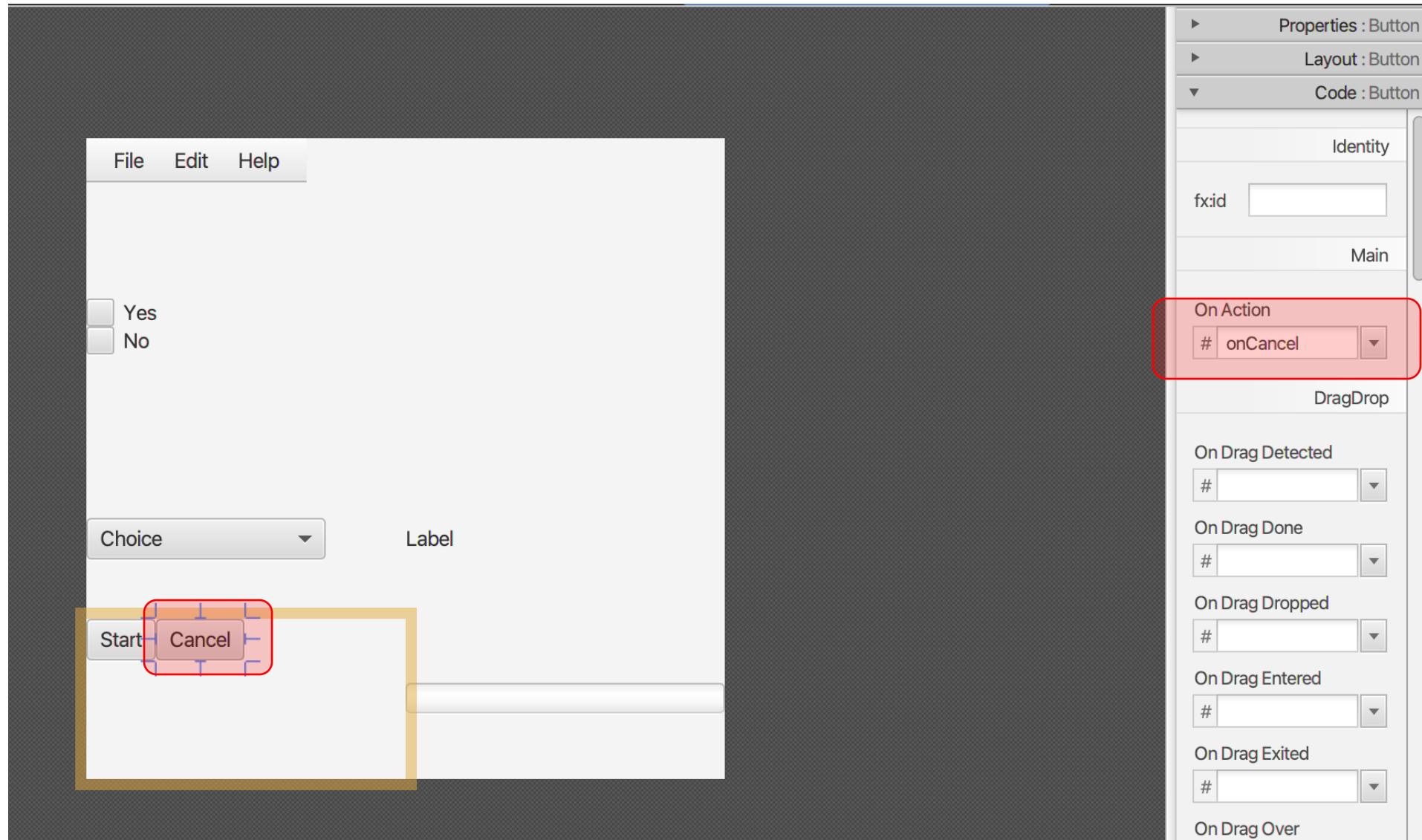
On Drag Dropped

On Drag Entered

On Drag Exited

On Drag Over

</>



The screenshot shows a user interface (UI) design application with a dark-themed interface. On the left is a main canvas area containing a window frame with a title bar labeled '0' and '1'. The window has a 'File' menu with 'Edit' and 'Help' options. Inside the window, there is a vertical stack of components: a 'Yes' button, a 'No' button, a 'Choice' dropdown menu, and a 'Start' button. Below these is a horizontal progress bar with a red highlight, also labeled '0' and '1'. To the right of the canvas is a vertical panel titled 'Properties : ProgressBar' which lists 'Layout : ProgressBar' and 'Code : ProgressBar'. At the bottom of this panel is an 'Identity' section with a red border containing the 'fx:id' field set to 'myProgressBar'. The right edge of the screen features a blue circular icon with the text '</>'.

Properties : ProgressBar

Layout : ProgressBar

Code : ProgressBar

Identity

fx:id myProgressBar

DragDrop

On Drag Detected

On Drag Done

On Drag Dropped

On Drag Entered

On Drag Exited

On Drag Over

On Mouse Drag Entered

On Mouse Drag Exited

16 </>

```
@FXML private ProgressBar myProgressBar;
```

</>

3 usages

17

```
private ProgressWorker pw=new ProgressWorker();
```

DEMO

59

```
@FXML
```

60

```
private void doWork(){
```

61

```
    new Thread(pw).start();
```

62

```
    myProgressBar.progressProperty().bind(pw.progressProperty());
```

63

```
}
```

16 </>

```
@FXML private ProgressBar myProgressBar;
```

</>

3 usages

17

```
private ProgressWorker pw=new ProgressWorker();
```

DEMO

59

```
@FXML
```

60

```
private void doWork(){
```

61

```
    new Thread(pw).start();
```

62

```
    myProgressBar.progressProperty().bind(pw.progressProperty());
```

63

```
}
```

65

```
@FXML
```

66

```
private void onCancel(){
```

67

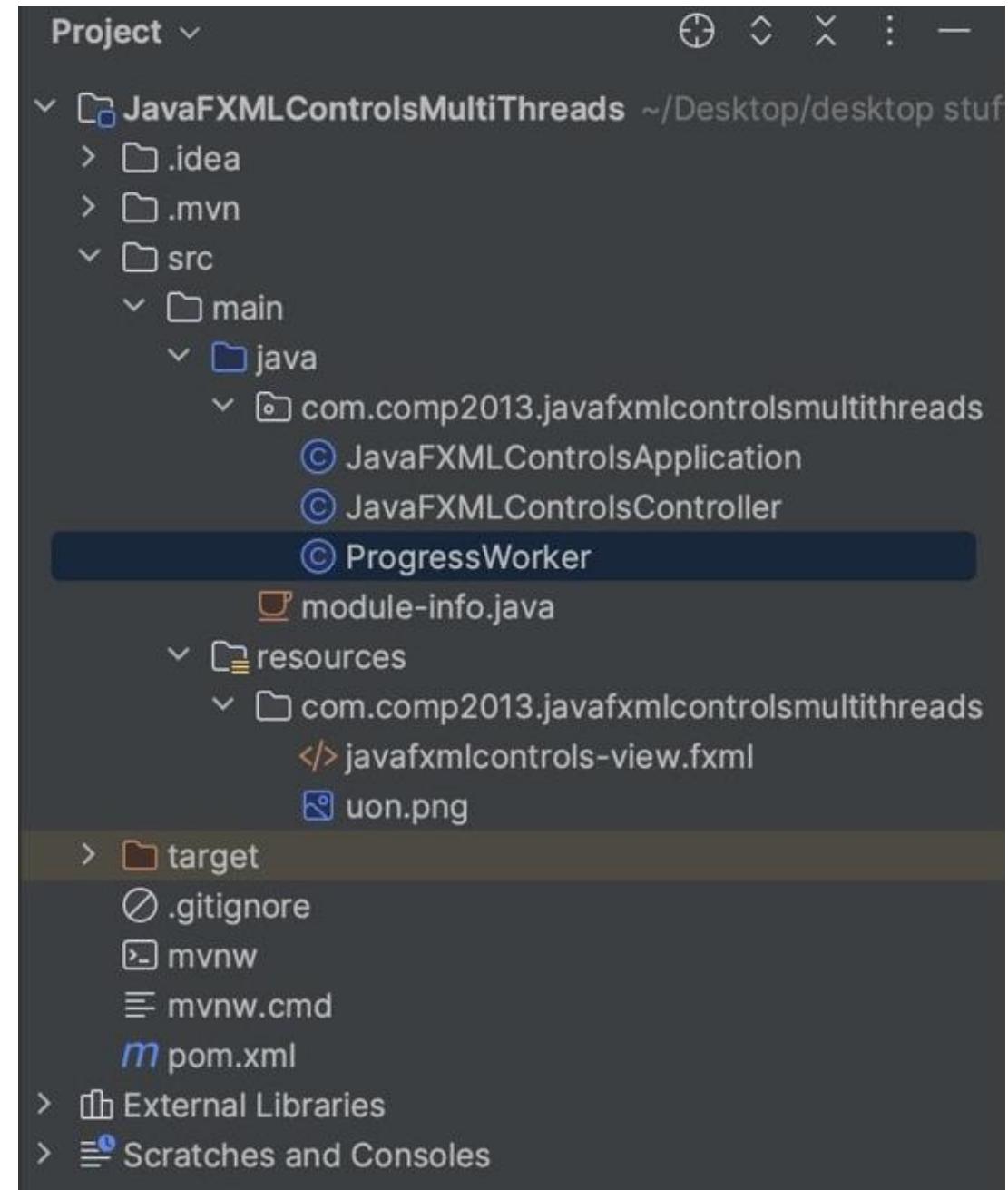
```
    pw.cancel();
```

68

```
}
```

</>

DEMO

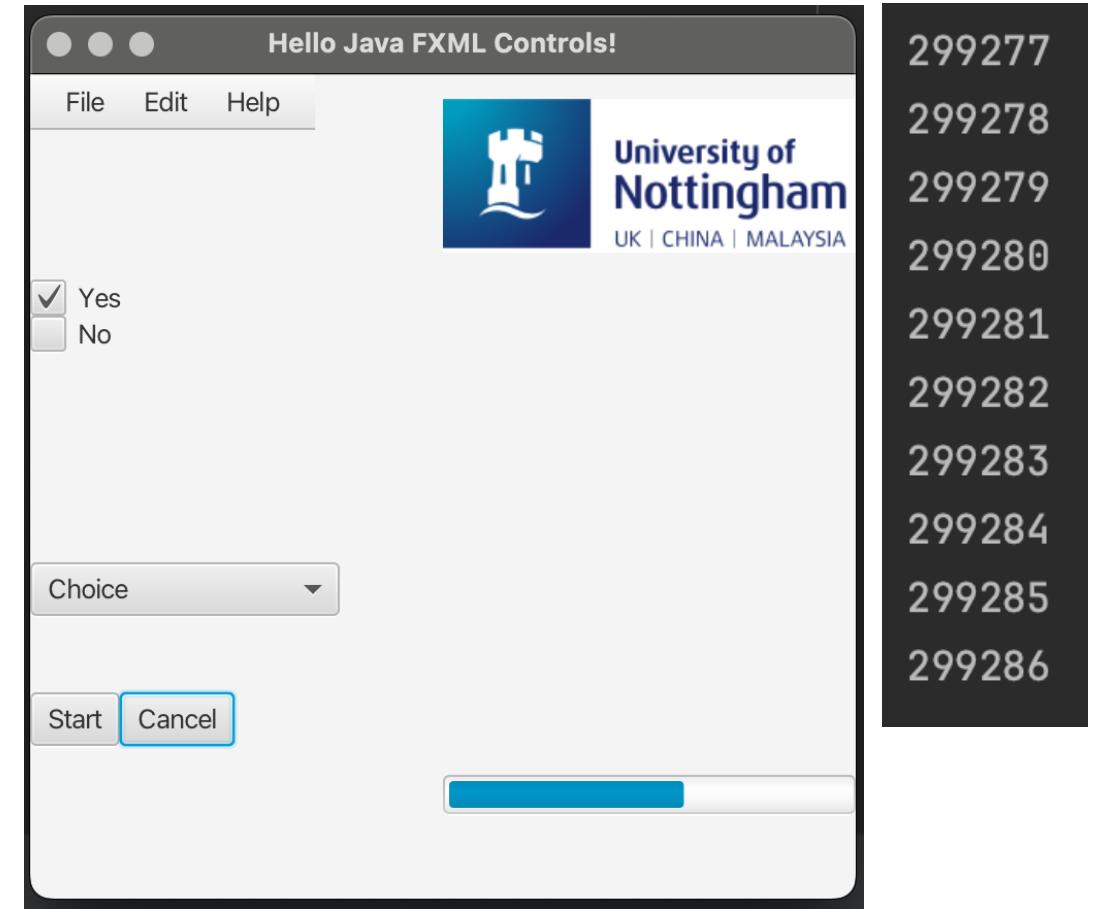
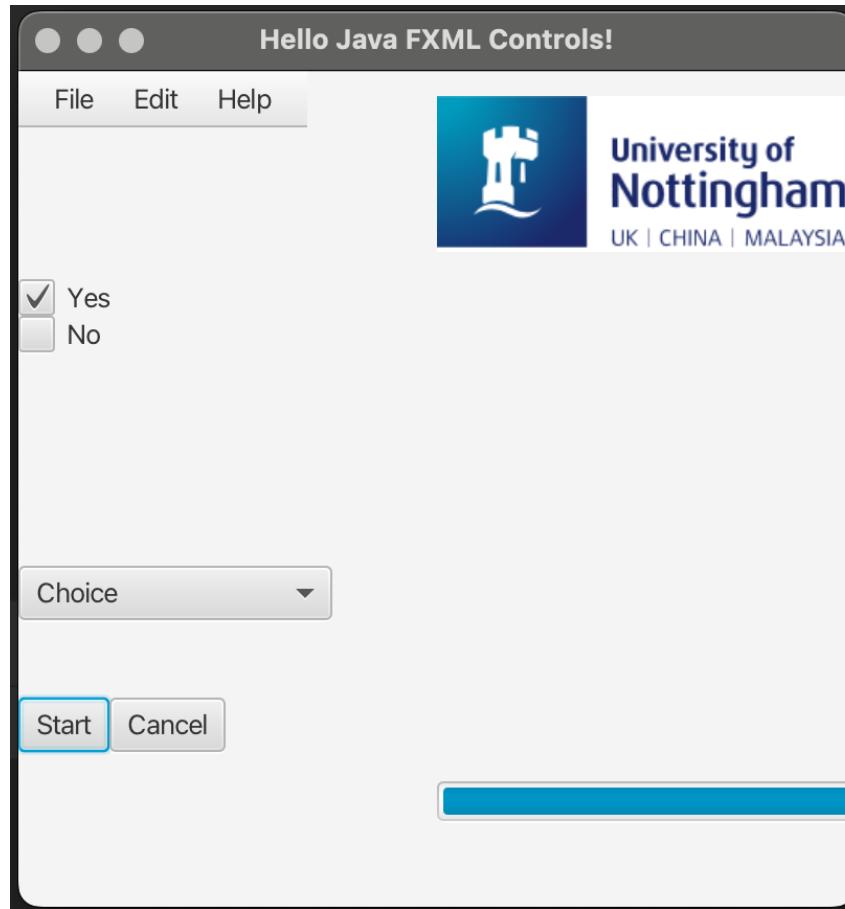


DEMO

```
1 package com.comp2013.javafxxmlcontrolsmultithreads;  
2  
3 import javafx.concurrent.Task;  
4  
5 public class ProgressWorker extends Task {  
6     @Override  
7     protected Object call() throws Exception {  
8         int n=500000;  
9         for(int i=0;i<n;i++){  
10             System.out.println(i);  
11             updateProgress(i,n);  
12             if(isCancelled())break;  
13         }  
14         return null;  
15     }  
16 }  
17 |
```

Result

</>



Concurrency in Java Advice

</>

<https://docs.oracle.com/javase/8/javafx/interoperability-tutorial/index.html>



Search Java SE Documentation

Looking for a different release? [Other releases](#)

Java Platform, Standard Edition (Java SE) 8

[Send Feedback](#) | [Print](#) | [PDF](#) | [ePub](#) | [Mobi](#)

JavaFX: Interoperability

Table of Contents

[Next Page](#)

[Expand](#) | [Collapse](#)

- Title and Copyright Information
- Preface
- Part I Concurrency in JavaFX
- Part II JavaFX-Swing Interoperability
- Part III Interoperability with SWT
- Part IV Source Code for the Interoperability Tutorial

JavaFX advanced topics

Animation

Types of animations in JavaFX

</>

- Timeline Animation
- Transition Animation
- Path Animation
- the list goes on :)

Key Concepts for Timeline Animation

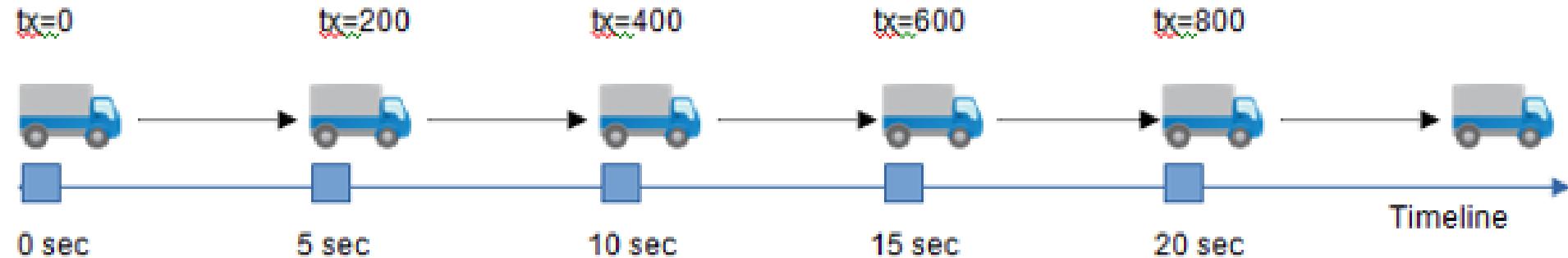
</>

- Timeline
 - Denotes the progression of time during animation with an associated key frame at a given instance
- Key frame
 - Represents the state of the node being animated at a specific instant on the timeline
- Key value
 - Represents the value of a property of a node along with an interpolator to be used
- Interpolator
 - By default linear; changes the property being animated linearly with time

Key Concepts for Timeline Animation

</>

- Lorries represent key frames at specific instants of the timeline
- The key values (here the value for the translateX property) associated with key frames are shown at the top
- By default linear interpolation is used between the key frames



Key Concepts for Timeline Animation

</>

DEMO

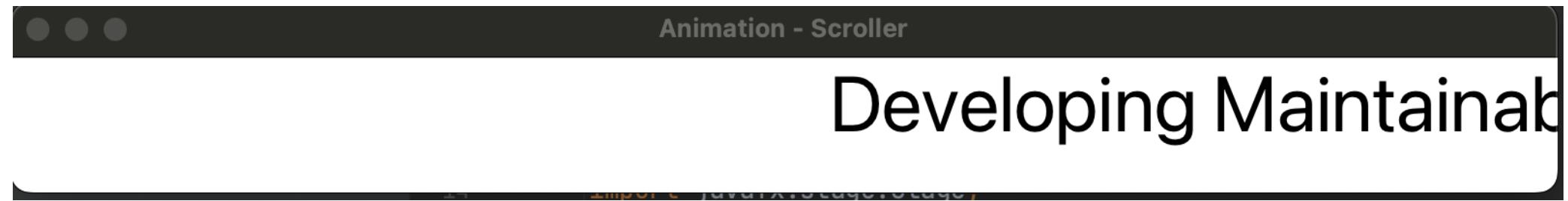
- Using timeline animation involves the following steps
 1. Construct key frames
 2. Create timeline object with key frames
 3. Set the animation properties
 4. Use the play() method to run the animation
- The timeline keeps all key frames in the ObservableList<KeyFrame> object
- The getKeyFrames() method returns the list

Key Concepts for Timeline Animation

</>

DEMO

- Let's look at an example



- Problem:
 - Scroll text does not update its initial position when the width of the scene changes
- Solution:
 - Update the initial key frame whenever the scene width changes
 - Use a ChangeListener for this



m pom.xml (JavaFXAnimation) ⌂ HelloApplication.java x module-info.java (JavaFXAnimation)

```
1 package com.javafxanimation;
2 > import ...
15 > public class HelloApplication extends Application {
16     @Override
17     public void start(Stage primaryStage) throws Exception{
18         Media sound = new Media(HelloApplication.class.getResource("AxelF.mp3").toString());
19         MediaPlayer mediaPlayer = new MediaPlayer(sound);
20         mediaPlayer.play();
21
22         Text msg=new Text("Developing Maintainable Software is Cool");
23         msg.setFont(Font.font(40));
24         msg.setTextOrigin(VPos.TOP);
25
26         Pane root=new Pane(msg);
27         root.setPrefSize(800, 70);
28         primaryStage.setTitle("Animation - Scroller");
29         Scene scene=new Scene(root);
30         primaryStage.setScene(scene);
31         primaryStage.show();
32
33         double sceneWidth=primaryStage.getWidth();
34         double msgWidth=msg.getLayoutBounds().getWidth();
35         KeyValue initKeyValue=new KeyValue(msg.translateXProperty(), sceneWidth); // -msgWidth
36         KeyFrame initFrame=new KeyFrame(Duration.ZERO, initKeyValue);
37         KeyValue endKeyValue=new KeyValue(msg.translateXProperty(), -msgWidth); // 0
38         KeyFrame endFrame=new KeyFrame(Duration.seconds(3), initKeyValue, endKeyValue);
39         Timeline timeline =new Timeline(initFrame, endFrame);
40         timeline.setCycleCount(Timeline.INDEFINITE);
41         timeline.setAutoReverse(true);
42         timeline.setRate(2);
43         timeline.play();
44     }
45
46     public static void main(String[] args) {
47         launch(args);
48     }
49 }
50 }
```

m pom.xml (JavaFXAnimation)

© HelloApplication.java ×

module-info.java (JavaFXAnimation)

```
18
19     @Override
20     public void start(Stage primaryStage) throws Exception{
21         Media sound = new Media(HelloApplication.class.getResource("AxelF.mp3").toString());
22         MediaPlayer mediaPlayer = new MediaPlayer(sound);
23         mediaPlayer.play();
24
25         Text msg=new Text("Developing Maintainable Software is Cool");
26         msg.setFont(Font.font(40));
27         msg.setTextOrigin(VPos.TOP);
28
29         Pane root=new Pane(msg);
30         root.setPrefSize(800, 70);
31         primaryStage.setTitle("Animation - Scroller");
32         Scene scene=new Scene(root);
33         primaryStage.setScene(scene);
34         primaryStage.show();
35
36         double sceneWidth=primaryStage.getWidth();
37         double msgWidth=msg.getLayoutBounds().getWidth();
38         KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); //msgWidth
39         KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);
40         KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); //0
41         KeyFrame endFrame=new KeyFrame(Duration.seconds(3),initKeyValue,endKeyValue);
42         Timeline timeline =new Timeline(initFrame,endFrame);
43         timeline.setCycleCount(Timeline.INDEFINITE);
44         timeline.setAutoReverse(true);
45         timeline.setRate(2);
46         timeline.play();
47     }
48
49     public static void main(String[] args) {
50
51         launch(args);
52     }
53
54 }
```

m pom.xml (JavaFXAnimation)

© HelloApplication.java ×

module-info.java (JavaFXAnimation)

```
18  
19 @Override  
20 public void start(Stage primaryStage) throws Exception{  
21     Media sound = new Media(HelloApplication.class.getResource(name: "AxelF.mp3").toString());  
22     MediaPlayer mediaPlayer = new MediaPlayer(sound);  
23     mediaPlayer.play();  
24  
25     Text msg=new Text(s: "Developing Maintainable Software is Cool");  
26     msg.setFont(Font.font(v: 40));  
27     msg.setTextOrigin(VPos.TOP);  
28  
29     Pane root=new Pane(msg);  
30     root.setPrefSize(v: 800, v1: 70);  
31     primaryStage.setTitle("Animation - Scroller");  
32     Scene scene=new Scene(root);  
33     primaryStage.setScene(scene);  
34     primaryStage.show();  
35  
36     double sceneWidth=primaryStage.getWidth();  
37     double msgWidth=msg.getLayoutBounds().getWidth();  
38     KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); // -msgWidth  
39     KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);  
40     KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); // 0  
41     KeyFrame endFrame=new KeyFrame(Duration.seconds(v: 3),initKeyValue,endKeyValue);  
42     Timeline timeline =new Timeline(initFrame,endFrame);  
43     timeline.setCycleCount(Timeline.INDEFINITE);  
44     timeline.setAutoReverse(true);  
45     timeline.setRate(2);  
46     timeline.play();  
47 }  
48  
49 public static void main(String[] args) {  
50     launch(args);  
51 }  
52 }  
53 }  
54 }
```

m pom.xml (JavaFXAnimation)

© HelloApplication.java ×

module-info.java (JavaFXAnimation)

```
18  
19 @Override  
20 @  
public void start(Stage primaryStage) throws Exception{  
    Media sound = new Media(HelloApplication.class.getResource("AxelF.mp3").toString());  
    MediaPlayer mediaPlayer = new MediaPlayer(sound);  
    mediaPlayer.play();  
      
    Text msg=new Text("Developing Maintainable Software is Cool");  
    msg.setFont(Font.font("V: 40"));  
    msg.setTextOrigin(VPos.TOP);  
      
    Pane root=new Pane(msg);  
    root.setPrefSize("V: 800, V1: 70");  
    primaryStage.setTitle("Animation - Scroller");  
    Scene scene=new Scene(root);  
    primaryStage.setScene(scene);  
    primaryStage.show();  
      
    double sceneWidth=primaryStage.getWidth();  
    double msgWidth=msg.getLayoutBounds().getWidth();  
    KeyValue initKeyValue=new KeyValue(msg.translateXProperty(), sceneWidth); // -msgWidth  
    KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);  
    KeyValue endKeyValue=new KeyValue(msg.translateXProperty(), -msgWidth); // 0  
    KeyFrame endFrame=new KeyFrame(Duration.seconds("V: 3"),initKeyValue,endKeyValue);  
    Timeline timeline =new Timeline(initFrame,endFrame);  
    timeline.setCycleCount(Timeline.INDEFINITE);  
    timeline.setAutoReverse(true);  
    timeline.setRate(2);  
    timeline.play();  
}  
public static void main(String[] args) {  
    launch(args);  
}
```

m pom.xml (JavaFXAnimation)

© HelloApplication.java ×

module-info.java (JavaFXAnimation)

```
18
19     @Override
20     public void start(Stage primaryStage) throws Exception{
21         Media sound = new Media(HelloApplication.class.getResource("AxelF.mp3").toString());
22         MediaPlayer mediaPlayer = new MediaPlayer(sound);
23         mediaPlayer.play();
24
25         Text msg=new Text("Developing Maintainable Software is Cool");
26         msg.setFont(Font.font(40));
27         msg.setTextOrigin(VPos.TOP);
28
29         Pane root=new Pane(msg);
30         root.setPrefSize(800, 70);
31         primaryStage.setTitle("Animation - Scroller");
32         Scene scene=new Scene(root);
33         primaryStage.setScene(scene);
34         primaryStage.show();
35
36         double sceneWidth=primaryStage.getWidth();
37         double msgWidth=msg.getLayoutBounds().getWidth();
38         KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); //msgWidth
39         KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);
40         KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); //0
41         KeyFrame endFrame=new KeyFrame(Duration.seconds(3),initKeyValue,endKeyValue);
42         Timeline timeline =new Timeline(initFrame,endFrame);
43         timeline.setCycleCount(Timeline.INDEFINITE);
44         timeline.setAutoReverse(true);
45         timeline.setRate(2);
46         timeline.play();
47     }
48
49     public static void main(String[] args) {
50
51         launch(args);
52     }
53
54 }
```

```

36     double sceneWidth=primaryStage.getWidth();
37     double msgWidth=msg.getLayoutBounds().getWidth();
38     KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); // -msgWidth
39     KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);
40     KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); // 0
41     KeyFrame endFrame=new KeyFrame(Duration.seconds( v: 3),initKeyValue,endKeyValue);
42     Timeline timeline =new Timeline(initFrame,endFrame);
43     timeline.setCycleCount(Timeline.INDEFINITE);
44     timeline.setAutoReverse(true);
45     timeline.setRate(2);
46     timeline.play();
47 }
48
49 public static void main(String[] args) {
50     launch(args);
51 }
52 }
53 }
54

```

```

18
19
20 @Override
21 public void start(Stage primaryStage) throws Exception{
22     Media sound = new Media(HelloApplication.class.getResource( name: "AxelF.mp3").toString());
23     MediaPlayer mediaPlayer = new MediaPlayer(sound);
24     mediaPlayer.play();
25
26     Text msg=new Text( s: "Developing Maintainable Software is Cool");
27     msg.setFont(Font.font( v: 40));
28     msg.setTextOrigin(VPos.TOP);
29
30     Pane root=new Pane(msg);
31     root.setPrefSize( v: 800, v1: 70);
32     primaryStage.setTitle("Animation - Scroller");
33     Scene scene=new Scene(root);
34     primaryStage.setScene(scene);
35     primaryStage.show();

```

```

36     double sceneWidth=primaryStage.getWidth();
37     double msgWidth=msg.getLayoutBounds().getWidth();
38     KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); // -msgWidth
39     KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);
40     KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); // 0
41     KeyFrame endFrame=new KeyFrame(Duration.seconds( v: 3),initKeyValue,endKeyValue);
42     Timeline timeline =new Timeline(initFrame,endFrame);
43     timeline.setCycleCount(Timeline.INDEFINITE);
44     timeline.setAutoReverse(true);
45     timeline.setRate(2);
46     timeline.play();
47 }
48
49 public static void main(String[] args) {
50     launch(args);
51 }
52 }
53 }
54

```

pom.xml (JavaFXAnimation) module-info.java (JavaFXAnimation)

```

18
19
20 @Override
21 public void start(Stage primaryStage) throws Exception{
22     Media sound = new Media(HelloApplication.class.getResource( name: "AxelF.mp3").toString());
23     MediaPlayer mediaPlayer = new MediaPlayer(sound);
24     mediaPlayer.play();
25
26     Text msg=new Text( s: "Developing Maintainable Software is Cool");
27     msg.setFont(Font.font( v: 40));
28     msg.setTextOrigin(VPos.TOP);
29
30     Pane root=new Pane(msg);
31     root.setPrefSize( v: 800, v1: 70);
32     primaryStage.setTitle("Animation - Scroller");
33     Scene scene=new Scene(root);
34     primaryStage.setScene(scene);
35     primaryStage.show();

```

```

36     double sceneWidth=primaryStage.getWidth();
37     double msgWidth=msg.getLayoutBounds().getWidth();
38     KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); // -msgWidth
39     KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);
40     KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); // 0
41     KeyFrame endFrame=new KeyFrame(Duration.seconds( v: 3),initKeyValue,endKeyValue);
42     Timeline timeline =new Timeline(initFrame,endFrame);
43     timeline.setCycleCount(Timeline.INDEFINITE);
44     timeline.setAutoReverse(true);
45     timeline.setRate(2);
46     timeline.play();
47 }
48
49 public static void main(String[] args) {
50     launch(args);
51 }
52 }
53 }
54

```

```

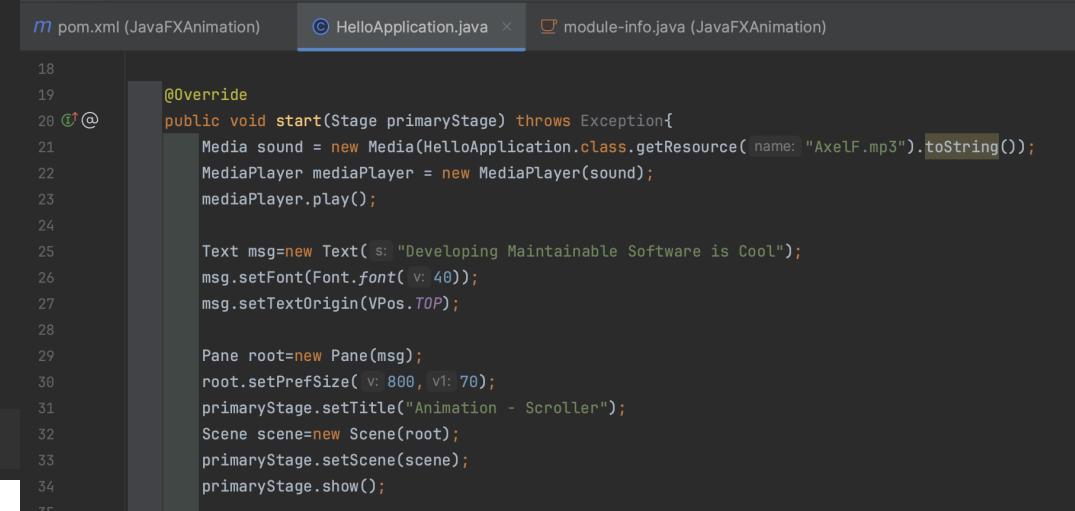
18
19
20 @Override
21 public void start(Stage primaryStage) throws Exception{
22     Media sound = new Media(HelloApplication.class.getResource( name: "AxelF.mp3").toString());
23     MediaPlayer mediaPlayer = new MediaPlayer(sound);
24     mediaPlayer.play();
25
26     Text msg=new Text( s: "Developing Maintainable Software is Cool");
27     msg.setFont(Font.font( v: 40));
28     msg.setTextOrigin(VPos.TOP);
29
30     Pane root=new Pane(msg);
31     root.setPrefSize( v: 800, v1: 70);
32     primaryStage.setTitle("Animation - Scroller");
33     Scene scene=new Scene(root);
34     primaryStage.setScene(scene);
35     primaryStage.show();

```

```

36         double sceneWidth=primaryStage.getWidth();
37         double msgWidth=msg.getLayoutBounds().getWidth();
38         KeyValue initKeyValue=new KeyValue(msg.translateXProperty(),sceneWidth); // -msgWidth
39         KeyFrame initFrame=new KeyFrame(Duration.ZERO,initKeyValue);
40         KeyValue endKeyValue=new KeyValue(msg.translateXProperty(),-msgWidth); // 0
41         KeyFrame endFrame=new KeyFrame(Duration.seconds( v: 3),initKeyValue,endKeyValue);
42         Timeline timeline =new Timeline(initFrame,endFrame);
43         timeline.setCycleCount(Timeline.INDEFINITE);
44         timeline.setAutoReverse(true);
45         timeline.setRate(2);
46         timeline.play();
47     }
48
49     public static void main(String[] args) {
50
51         launch(args);
52     }
53 }
54

```



```

m pom.xml (JavaFXAnimation) C HelloApplication.java module-info.java (JavaFXAnimation)
18
19
20 @Override
21 public void start(Stage primaryStage) throws Exception{
22     Media sound = new Media(HelloApplication.class.getResource( name: "AxelF.mp3").toString());
23     MediaPlayer mediaPlayer = new MediaPlayer(sound);
24     mediaPlayer.play();
25
26     Text msg=new Text( s: "Developing Maintainable Software is Cool");
27     msg.setFont(Font.font( v: 40));
28     msg.setTextOrigin(VPos.TOP);
29
30     Pane root=new Pane(msg);
31     root.setPrefSize( v: 800, v1: 70);
32     primaryStage.setTitle("Animation - Scroller");
33     Scene scene=new Scene(root);
34     primaryStage.setScene(scene);
35     primaryStage.show();

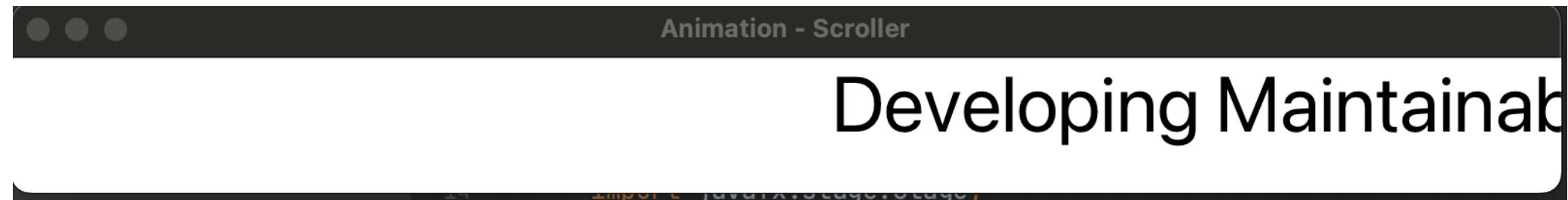
```

Key Concepts for Timeline Animation

</>

DEMO

- Let's look at an example



- Problem:
 - Scroll text does not update its initial position when the width of the scene changes
- Solution:
 - Update the initial key frame whenever the scene width changes
 - Use a ChangeListener for this

- Challenge: Multi-threading
 - Modify the "makeCall()" method to run in another thread, emulating a 10 second call. We do not want the phone to "lock up" while we are making a call.
- Challenge: Animation
 - When the user clicks call, add a suitable animation on the front of the interface that represents a call being connected
 - Use the Timeline and KeyFrame classes

- General JavaFX
 - Comprehensive introduction to JavaFX
 - https://www3.ntu.edu.sg/home/ehchua/programming/java/javafx1_intro.html
 - Multithreading & Concurrent Programming in Java
 - https://www3.ntu.edu.sg/home/ehchua/programming/java/J5e_multithreading.html
- Animation of Text in JavaFX
 - <https://examples.javacodegeeks.com/desktop-java/javafx/javafx-animation-example-2/>

some final remarks ...



University of
Nottingham

UK | CHINA | MALAYSIA

Lecture 7—Design Principles and Design Patterns

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior



Register your attendance

COMP2013: Developing Maintainable Software
Week 8 – 4:00pm Monday – 13 November 2023



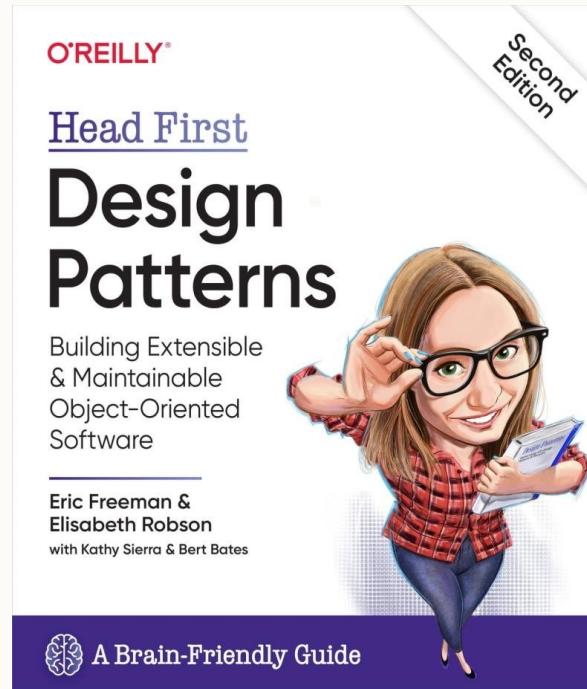
valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14



COMP2013: Developing Maintainable Software
Week 8 – 4:00pm Monday – 13 November 2023



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



Overview

- Q/A with a 3rd Year Student
- Generating automatic class diagram
- SOLID principles
- Design Patterns



Topics for this Week

- Lecture 07A:
 - Design Principles and Design Patterns
- Lecture 07B:
 - Continuation of 07A
 - Review of some exam related questions
- Lab 07:
 - Continue with your coursework



From Concepts to Patterns

Design Patterns and SOLID Principles extend the Object Oriented Philosophy



From Concepts to Patterns

- OO Concepts
 - Object model (abstraction, encapsulation, modularity, hierarchy); data abstraction; inheritance; polymorphism; interfaces
- OO Design Principles
 - Encapsulate what varies; favour object aggregation over class inheritance; program to interfaces not implementations;
- OO Design Patterns:
 - Show how to build systems with good OO design qualities (reusable; extensible; maintainable)



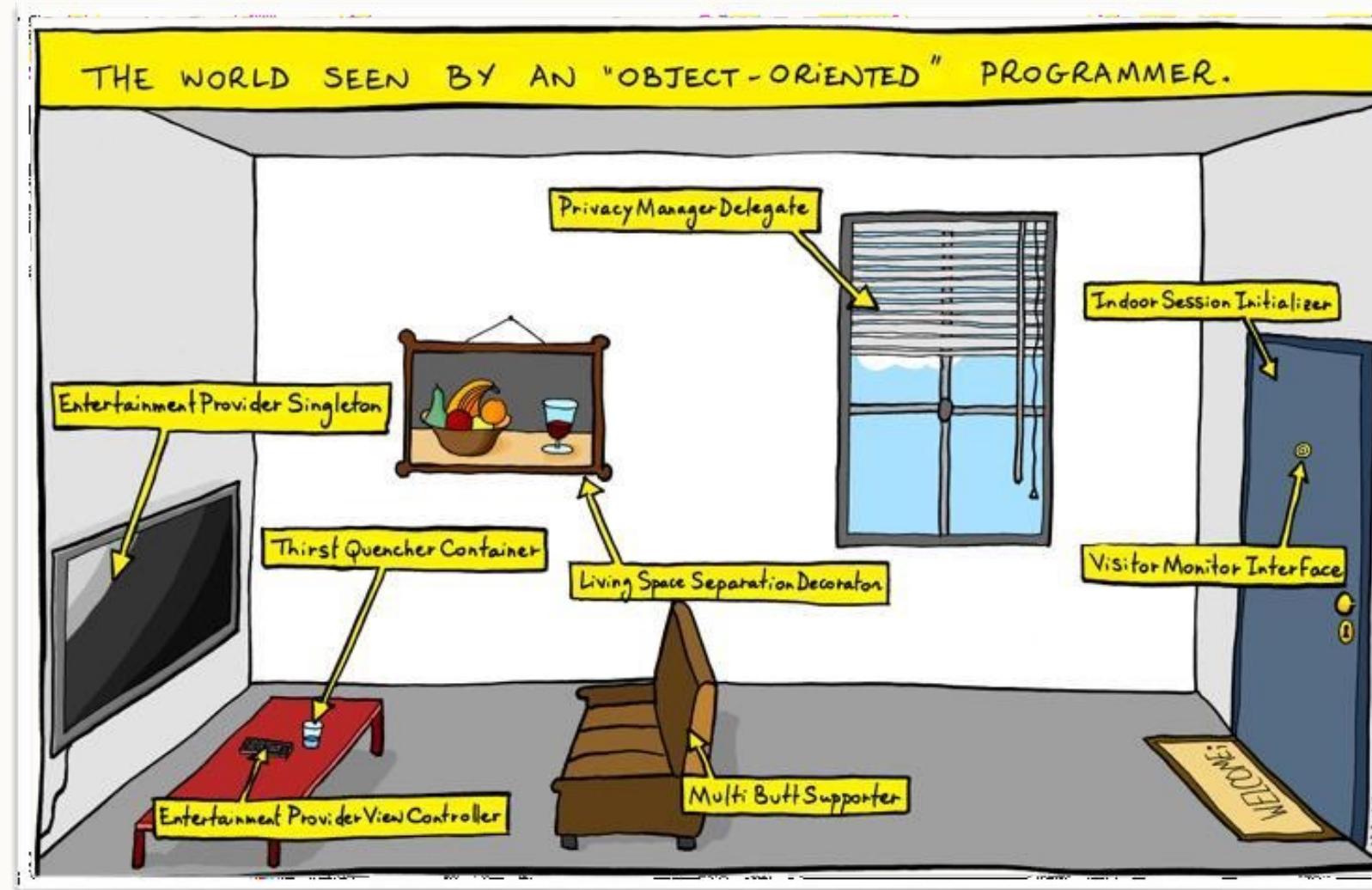
Why OOD Principles and Patterns

- Software development specifications keep changing constantly.
- Your software solves real life problems and processes in real life (and your knowledge) evolve.
 - This is in particular true in the business world.
- Your software need to be smart enough to cope.
 - Easy to maintain (by applying changes with minimal effort)
 - Easy to extend (without changing existing code)
- Following systematically conventions so others can also maintain your code.



You don't always have to Object Orientate, but when you do...

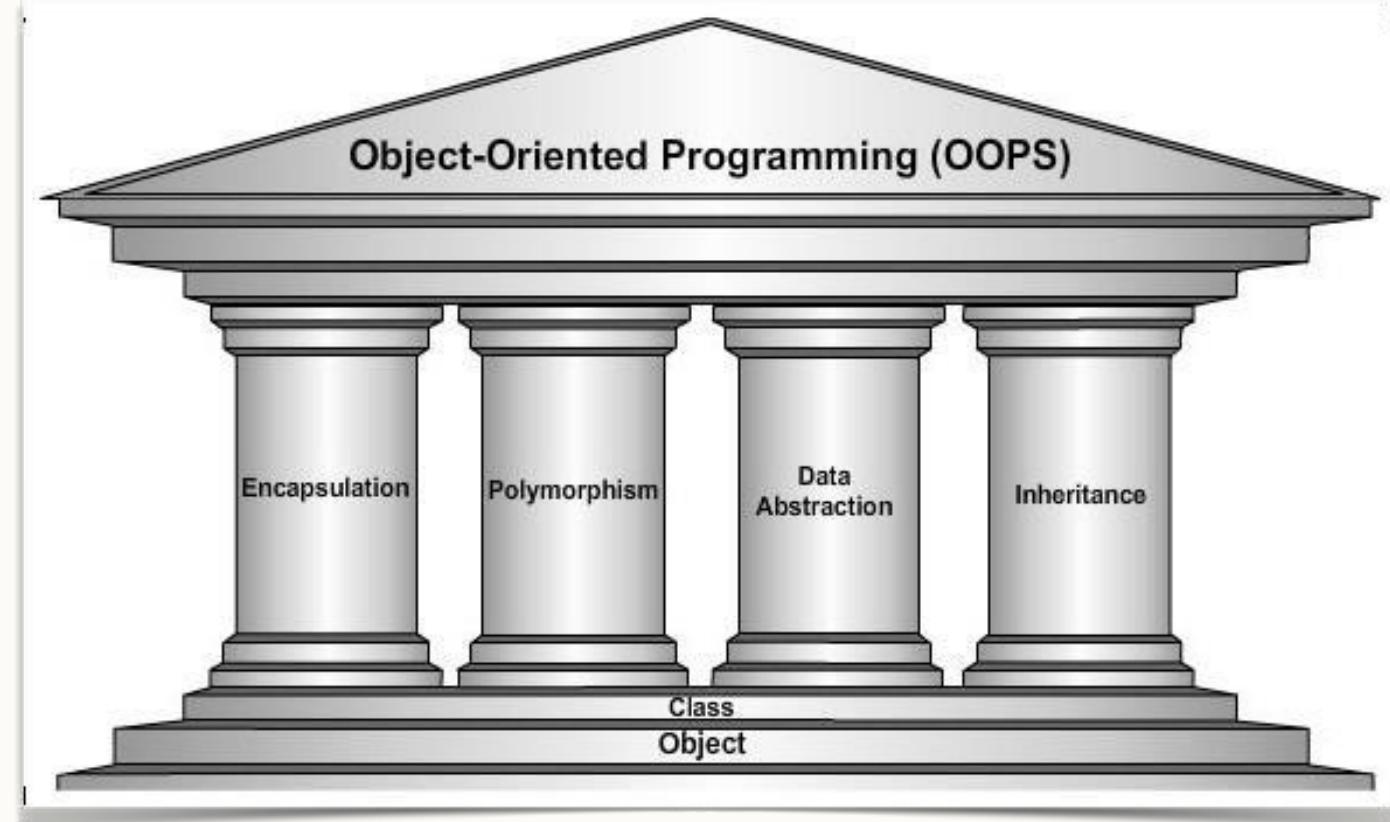
The Bottom Line: Object Orientation





The Pillars of Object Orientation

- Abstraction
- Inheritance
- Encapsulation
- Polymorphism





The Object Model [Booch 1994]

- Abstraction
 - Denotes the **essential characteristics of an object** that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.
 - Concentrating only on essential characteristics.
- Encapsulation:
 - Process of **compartmentalising the elements of an abstraction** that constitute its structure and behaviour.
 - Encapsulation serves to separate the contractual interface of an abstraction and its implementation.



The Object Model [Booch 1994]

- Modularity:
 - Modularity is the property of a system that has been **decomposed** into a set of **cohesive** and **loosely coupled modules** (classes).
 - Cohesive: Performing a single type of tasks
 - Loosely coupled: Highly independent
- Hierarchy:
 - A **ranking or ordering of abstractions**
 - Classes at the same level of the hierarchy should be at the same level of abstraction.



Core O-O Concepts: Abstraction and Inheritance

- Data Abstraction
 - Application of implementation hiding – you don't need to know all the details of how something works in order to make use of it.
 - Creating **new data types that are well suited to an application**; allows the creation of user defined data types, having the properties of build in data types and a set of permitted operators.
 - *Abstract Data Type*: A structure that contains both, data and the definition of actions to be performed with that data where a **class** is an implementation of an abstract data type.



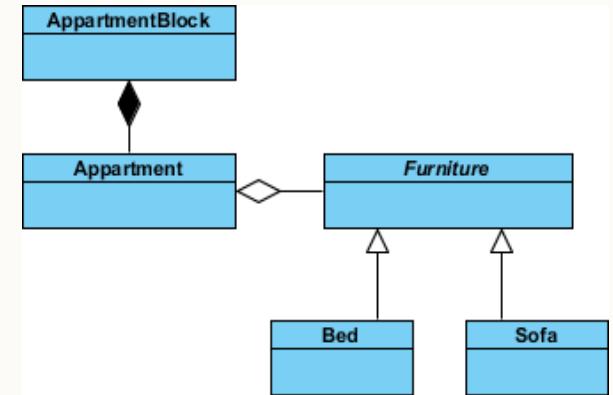
Core O-O Concepts: Abstraction and Inheritance

- Inheritance
 - Inheritance defines **implementation** of a class in terms of another's
 - New data types can be defined as extensions to previously defined types
 - This allows reuse since the implementation is not repeated



Other Object Oriented Concepts

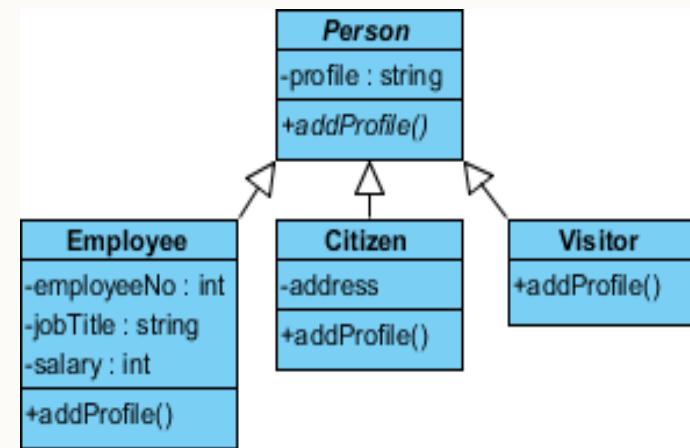
- Composition vs Aggregation – two forms of specialised containment
 - **Composition**
 - A relationship where a child cannot exist independent of the parent e.g., an apartment cannot exist outside of the ApartmentBlock
 - Implies that the **internal objects are not seen from the outside**
 - **Aggregation**
 - Implies that the child can exist without the parent e.g., the bed can exist outside of the apartment.
 - Obtaining new functionality by **assembling objects**
 - This allows reuse as aggregated objects have a separate existence.
 - Aggregated objects might be directly accessed.





Core O-O Concept: Abstract vs Concrete Classes

- Abstract class
 - A class that **cannot be directly instantiated**
 - An abstract class is written with the expectation that its concrete subclasses will add to its structure and behaviour.
 - Denote by marking classes and methods as abstract
- Concrete class
 - A class that can be **directly instantiated**
 - A class whose implementation is complete and thus may have instances.
- Still confused? Try this:
<https://www.youtube.com/watch?v=hZ1EU-F-OnU>





Core O-O Concept: Polymorphism

- This is treating different subclasses as the same superclass.
- Polymorphism through method **overloading** (design time polymorphism)
 - Create more than one function with same name but different signatures.
- Polymorphism through method **overriding** (run time polymorphism)
 - Create a function in the derived class with the same name and signature.



Core O-O Concept: Polymorphism

- Polymorphism through **sub classing** (run time polymorphism)
 - A reference variable of a base class is able to reference, instantiate and destroy objects of a derived class.
 - A reference variable may reference objects of many classes at runtime, but the compiler cannot predict which they will be at compile time.
- Still confused? Try this: <https://www.youtube.com/watch?v=SwEzCBM7n-Q>



Core O-O Concept: Interfaces

- In most cases interfaces are only method declarations and do not implement any of the methods.
- An interface **describes the behavior or capabilities of a class without committing to a particular implementation** of that class.
- An interface is a contract of related services and a set of conditions that must be true for the contract to be faithfully executed.
- Interfaces **formalise polymorphism**, they allow us to define polymorphism in a declarative way unrelated to implementation.
- Still confused? Try this: <https://www.youtube.com/watch?v=fX1xNMBTMfg>



Concurrency Framework in JavaFX

- Framework consist of one interface + four classes + one enum
 - **Worker** interface specifies the methods available to background (work) threads
 - **Task** class instance represents a one-shot task
 - **Service** class instance represents a reusable task
 - **ScheduledService** class instance represents a reusable task that runs repeatedly following a specified interval
 - **WorkerStateEvent** class instance represents an event that occurs as state of Worker changes
 - You can add event handlers to all three types of tasks to listen to the changes in their states
 - **State** enum constants represents different states of a worker



Self test: Things you should be able to define by now

Inheritance is...

Polymorphism is...

Abstraction is...

Encapsulation is...

An interface is...

Concrete classes are ...

Abstract classes are ...



SOLID Principles

Object Oriented Design Principles for Clean Coders



Uncle Bob

- Robert C. Martin aka ‘Uncle Bob’
- Agile and OOP guru
- Derive principles to improve the state of the art of *software craftsmanship*
 - *“Writing clean code is what you must do in order to call yourself a professional. There is no reasonable excuse for doing anything less than your best.”*





“SOLID” Design Principles and Clean



- Software solves real life business problems and real life business processes evolve and change - always.
- A smartly designed software can adjust changes easily; it can be extended, and it is re-usable.
- SOLID Principles (by Uncle Bob)
[\[http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod\]](http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod)
 - **S** = Single Responsibility Principle
 - **O** = Open-Closed Principle
 - **L** = Liskov's Substitution Principle
 - **I** = Interface Segregation Principle
 - **D** = Dependency Inversion Principle



Code Sense



- "A programmer without code-sense can look at a messy module and recognise the mess but will have no idea what to do about it"
- "A programmer with code-sense will look at a messy module and see options and variations. The code-sense will help the programmer chose the best variation and guide him or her to plot a sequence of behaviour-preserving transformations to get from here to there."
- SOLID principles help us to develop our **code sense**
- As with all design principles and patterns, they do not involve code reuse.
 - they “**employ experience reuse**”



</>

Single Responsibility Principle

Open Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle



Single Responsibility Principle

</>



SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

- UB says: "A class should have one and only one responsibility"



Single Responsibility Principle



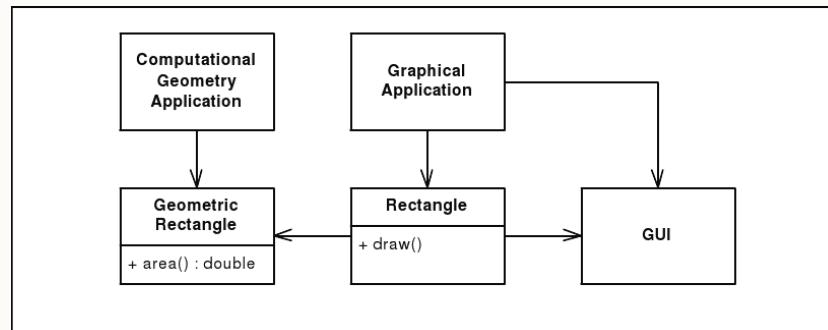
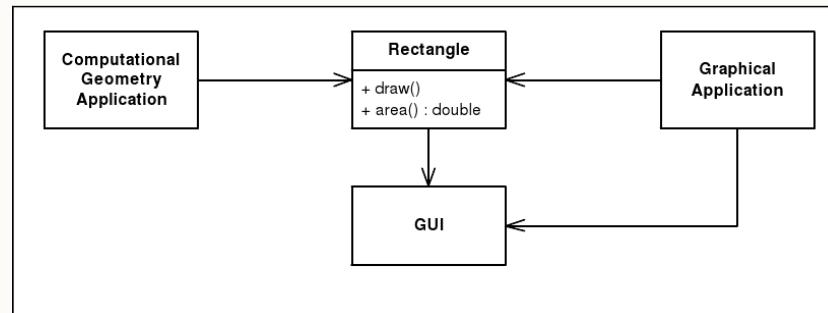
- UB says: "A class should have one and only one responsibility"
 - Code becomes coupled if classes have more than one responsibility.
 - Modifications done carry the risk of affecting other parts of the class
 - Classes with more than one responsibility need to be split up.
 - Supports reuse
 - Supports modifying minimal amount of code
- A method should have one and only one responsibility.
 - Methods should be broken down so that they do only one job.
 - Supports reuse
 - Supports modifying minimal amount of code



Single Responsibility Principle

</>

- A classic example from Uncle Bob





Single Responsibility Principle

- Here's an example of a Java class that does not follow the single responsibility principle (SRP):

```
public class Vehicle {  
    public void printDetails() {}  
    public double calculateValue() {}  
    public void addVehicleToDB() {}  
}
```

- By applying SRP, we can separate the above class into three classes with separate responsibilities.



Open-Closed Principle

</>



- "Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."



Open-Closed Principle



- "Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."
 - Open for Extension
 - The behavior of the module can be extended. That we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications.
 - Closed for Modification
 - The source code of such a module **is inviolate**. No one is allowed to make source code changes to it.

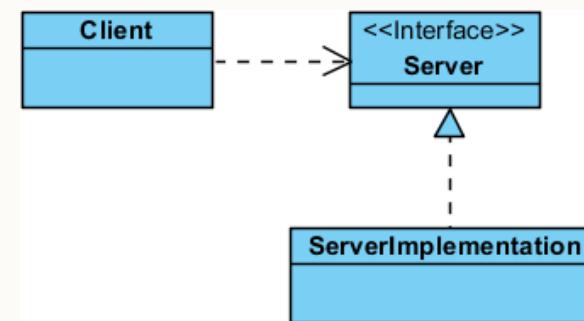
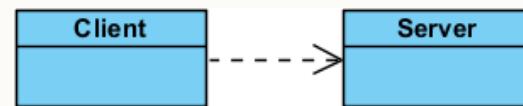
"Abstraction" is the Key



Open-Closed Principle

</>

- A classic example from Uncle Bob making good use of interfaces





Open-Closed Principle

</>

- Consider the below method of the class **VehicleCalculations**.
- A better approach would be for the subclasses **Car** and **Truck** to override the **calculateValue** method.

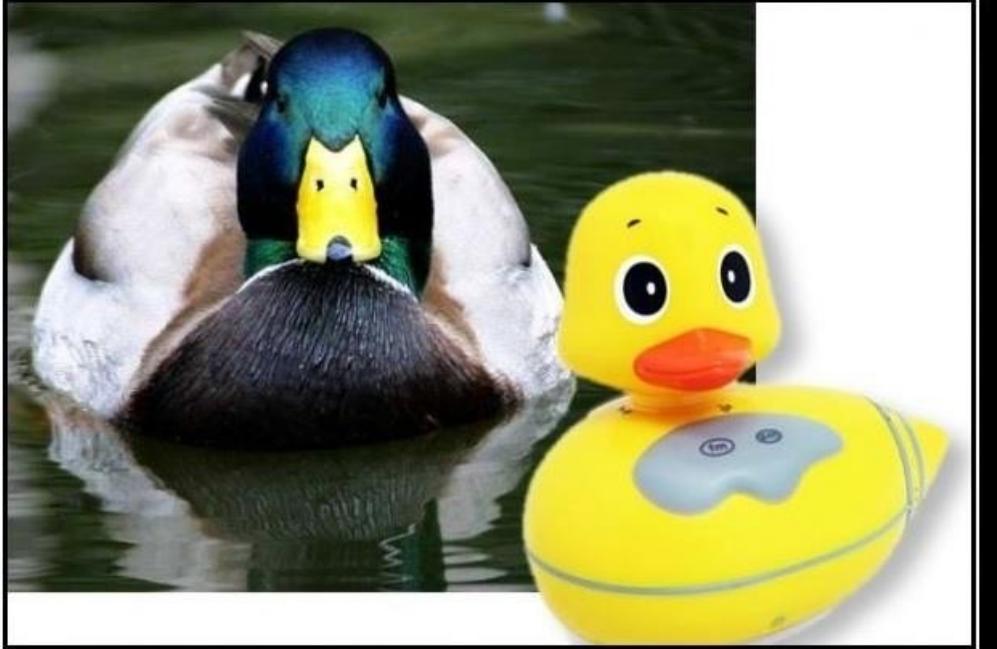
```
public class VehicleCalculations {  
    public double calculateValue(Vehicle v) {  
        if (v instanceof Car) {  
            return v.getValue() * 0.8;  
        if (v instanceof Bike) {  
            return v.getValue() * 0.5;  
        }  
    }  
}
```

```
public class Vehicle {  
    public double calculateValue() {...}  
}  
public class Car extends Vehicle {  
    public double calculateValue() {  
        return this.getValue() * 0.8;  
    }  
}  
public class Truck extends Vehicle{  
    public double calculateValue() {  
        return this.getValue() * 0.9;  
    }  
}
```



Liskov's Substitution Principle

</>



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You
Probably Have The Wrong Abstraction

- UB says: ""Subtypes must be substitutable for their base types."



Liskov's Substitution Principle

</>

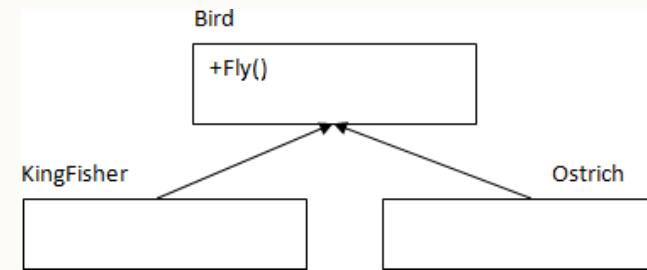
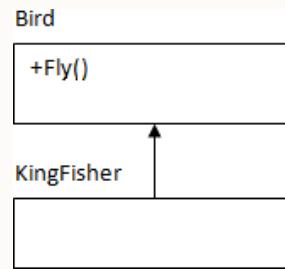
- UB says "Subtypes must be substitutable for their base types."
 - Methods that use references to base classes must be able to use objects of derived classes without knowing it.
 - The "is a" technique of determining inheritance relationships is simple and useful, but occasionally results in bad use of inheritance.

*This is just a way of
ensuring that "inheritance" is
used correctly*



Liskov's Substitution Principle

- A classic example from Uncle Bob by thinking about the functionality of classes.



- There should be a separate class for birds that can't really fly and Ostrich should inherit that.
- The subclass needs to be able to properly use the methods of the superclass, else the abstraction is wrong



Liskov's Substitution Principle

- The below classes violated the Liskov substitution principle because the Square class has extra constraints i.e., height and weight that must be the same.

```
public class Rectangle {  
    private double height;  
    private double width;  
    public void setHeight(double h) { height = h; }  
    public void setWidth(double w) { width = w; }  
    ...  
}  
public class Square extends Rectangle {  
    public void setHeight(double h) {  
        super.setHeight(h);  
        super.setWidth(h);  
    }  
    public void setWidth(double w) {  
        super.setHeight(w);  
        super.setWidth(w);  
    }  
}
```



Interface Segregation Principle

</>



- UB says: "Clients should not be forced to depend upon interfaces that they do not use."



Interface Segregation Principle



- UB says "Clients should not be forced to depend upon interfaces that they do not use."
 - If a class wants to implement the interface it has to implement all the methods.
 - Fat interfaces should be broken down.
 - The principle ensures that interfaces are developed so that each of them have their own responsibility and thus they are specific, easily understandable and reusable.
 - This is also an extension of the *single responsibility principle*.
 - Nice example:
<https://dzone.com/articles/solid-principles-interface-segregation-principle>





Interface Segregation Principle



- A classic example from Uncle Bob



- A SeaGull would implement the IFlyingBird interface
- An Penguin would implement the IBird interface



Interface Segregation Principle



- As you can see, it does not make sense for a Bike class to implement the openDoors() method as a bike does not have any doors!

```
public interface Vehicle {  
    public void drive();  
    public void stop();  
    public void refuel();  
    public void openDoors();  
}  
public class Bike implements Vehicle {  
  
    // Can be implemented  
    public void drive() {...}  
    public void stop() {...}  
    public void refuel() {...}  
  
    // Can not be implemented  
    public void openDoors() {...}  
}
```



Dependency Inversion Principle

</>



DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

- UB says: "High-level modules should not depend on low-level modules. Both should depend on abstractions.
Abstractions should not depend on details.
Details should depend on abstractions."



Dependency Inversion Principle

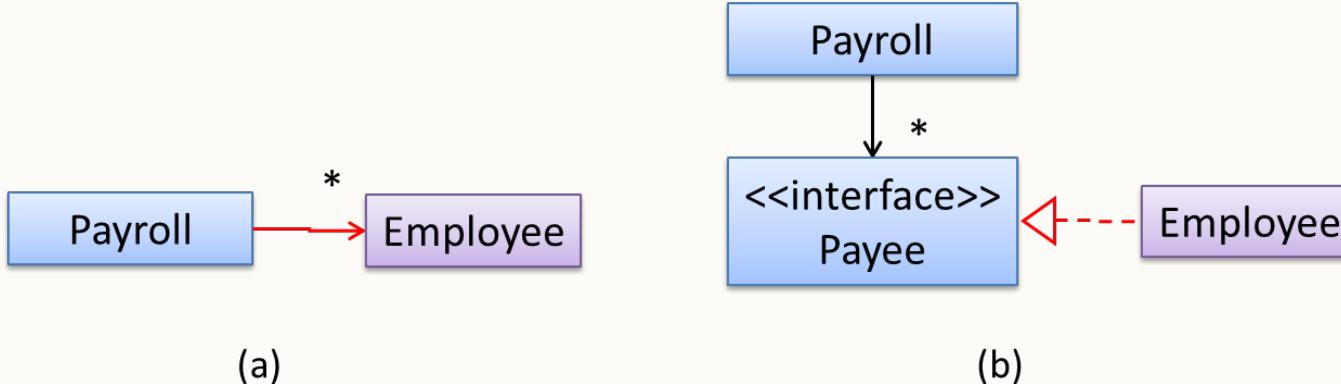


- UB says "High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions."
 - Goal is to reduce coupling between different pieces of code by adding a layer of abstraction.
 - Make the higher-level modules depend on abstractions rather than concrete implementation of lower-level modules.
 - This for me is the key principle in SOLID otherwise maintainable code is not achievable.

*Dependency inversion is a generalisation for the
Open-Close and Liskov Substitution Principles*



Dependency Inversion Principle



In design (a), the higher-level class **Payroll** depends on the lower-level class **Employee**, which is a violation of DIP. In design (b), both **Payroll** and **Employee** depend on the **Payee** interface (note that inheritance is a dependency).

Design (b) is more flexible (and less coupled) because now the **Payroll** class need not change when the **Employee** class changes.



Dependency Inversion Principle

- Consider the example below. We have a Car class that depends on the concrete Engine class; therefore, it is not obeying DIP.
- The code will work, for now, but what if we wanted to add another engine type, let's say a diesel engine? This will require refactoring the Car class.

```
public class Car {  
    private Engine engine;  
    public Car(Engine e) {  
        engine = e;  
    }  
    public void start() {  
        engine.start();  
    }  
}  
public class Engine {  
    public void start() {...}  
}
```



Dependency Inversion Principle



- Instead of Car depending directly on Engine, let's add an interface.
- Now we can connect any type of Engine that implements the Engine interface to the Car class.

```
public interface Engine {  
    public void start();  
}
```

```
public class Car {  
    private Engine engine;  
    public Car(Engine e) {  
        engine = e;  
    }  
    public void start() {  
        engine.start();  
    }  
}  
public class PetrolEngine implements Engine {  
    public void start() {...}  
}  
public class DieselEngine implements Engine {  
    public void start() {...}  
}
```



Brace Yourselves, Design Patterns Are Coming...



‘Gang of Four’ Design Patterns

Promoting experience reuse



Design Patterns formalise and extend O-O principles

- What is a Design Pattern?
 - A pattern describes a problem which occurs over and over again and then **describes the core of the solution** to that problem in such a way that it can use the solution over and over again without ever doing it the same way again.
 - A pattern provides an **abstract description** of a design problem and how a general arrangement of elements solves it.
 - A design pattern **identifies the participating classes and instances**, their roles and collaborations, and the distribution of responsibilities.
- A design pattern exists in code, not in your brain.



Design Patterns



A pattern specification has four essential elements:

1. **Name:** A handle that we can use to describe a design problem, its solution and the consequences in one or two words; having a vocabulary for patterns lets us talk about them with our colleagues and in our documentation.
2. **Problem:** Describes when to apply a pattern; sometimes the problem includes a list of conditions that must be met before it makes sense to apply a pattern.
3. **Solution:** Describes the elements that make up the design and their relationships.
4. **Consequences:** Describe the results and trade-offs (time and space) of applying the pattern; critical for evaluating design alternatives.



Design Patterns



- Design patterns are organised in two ways:
 - **Purpose:** Reflects what the pattern does
 - **Creational:** Concern the process of object creation.
 - **Structural:** Deal with the composition of classes and objects.
 - **Behavioural:** Characterise the way in which classes and objects interact and distribute responsibility.
 - **Scope:** Specifies whether the pattern applies to classes or objects
 - **Class:** These patterns deal with relationships between classes and sub-classes (**which are fixed at compile time**)
 - **Object:** Deal with object relationships (**which can be changed at runtime**)



		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor



Just Enough Design Patterns To Get By

</>

- How many design patterns do you really need to know?
 - The most useful should be known intimately;
 - The lesser used should be known well enough to be able to discuss them (even if implementation requires a quick refresher)
 - The remaining can be postponed until relevant (although not forgotten completely)
- Most useful design patterns
 - Adapter; Command; Decorator; Facade; Factory; Abstract Factory; Observer;
 - Strategy
 - State (this is Peer's opinion – works on agent-based simulation)
 - Factory and Singleton (in Julie's opinion – works on sensor data in cybersecurity)



Design Pattern Examples

http://www.tutorialspoint.com/design_pattern

Design Pattern Categories

- The Gang of Four produced twenty-three patterns which are either:
 - Creational
 - Structural
 - Behavioural
- In this lecture we will look at some of the most commonly used patterns

</>

The Sacred Elements of the Faith

the holy origins

107 FM Factory Method	117 PT Prototype	127 S Singleton
87 AF Abstract Factory	325 TM Template Method	233 CD Command
97 BU Builder	315 SR Strategy	283 MM Memento

the holy structures

139 A Adapter	223 CR Chain of Responsibility	163 CP Composite	175 D Decorator
243 IN Interpreter	207 PX Proxy	185 FA Façade	
331 V Visitor	195 FL Flyweight	151 BR Bridge	

the holy behaviors



Creational Patterns



Abstract factory: interface to create related objects without declaring the concrete class

Builder: a Builder class builds the final object step by step

Factory Method: defers instantiation to subclasses

Prototype: implementing a prototype interface which tells to create a clone of the current object

Singleton: ensures that one and only one instance of a class is created



Structural Patterns



Adapter: provides compatible interfaces for classes that couldn't work together otherwise

Bridge: involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes

Composite: put objects into a tree structure to represent the hierarchies

Decorator: add new functionality to an existing object without altering its structure

Facade: involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes

Flyweight: use sharing to support large numbers of complex objects

Proxy: provide a placeholder for another object to access its controls



Behavioural Patterns - 1

Chain of responsibility: give more than one object the chance to handle a request

Command: encapsulate a request as an object

Interpreter: convert problems expressed in natural language into a representation

Iterator: access to objects without exposing underlying representation

Mediator: promotes loose coupling by preventing objects from referring to each other



Behavioural Patterns -2



Memento: Capture and display an object's internal state

Observer: define a one to many relationship so that when one object changes state, all its dependents are notified

State: allow an object to alter its behaviour when its internal state changes.

Strategy: Define and encapsulate a family of algorithms. Let the algorithm vary independently of who is using it

Template Method: Define the skeleton of an algorithm, and let sub classes redefine certain steps without changing the structure of the algorithm

Visitor: define a new operation without changing the classes



The Singleton

“Create a one of a kind object for which there is only one instance”



Wow, this sounds really simple?

</>

- Has a single class in its class diagram
 - *BUT how do we ensure that there is **only** one instance??*
- To ensure a class only has one instance, and provide a global point of access to it
 - By using static methods and private classes
- Sometimes we want just a single instance of a class to exist in the system.
 - we want just one window manager or print spooler
- We need to have that one instance easily accessible, **and** we want to ensure that additional instances of the class can not be created.
- Example : <https://www.youtube.com/watch?v=8vOLNHMjgXM>



</>

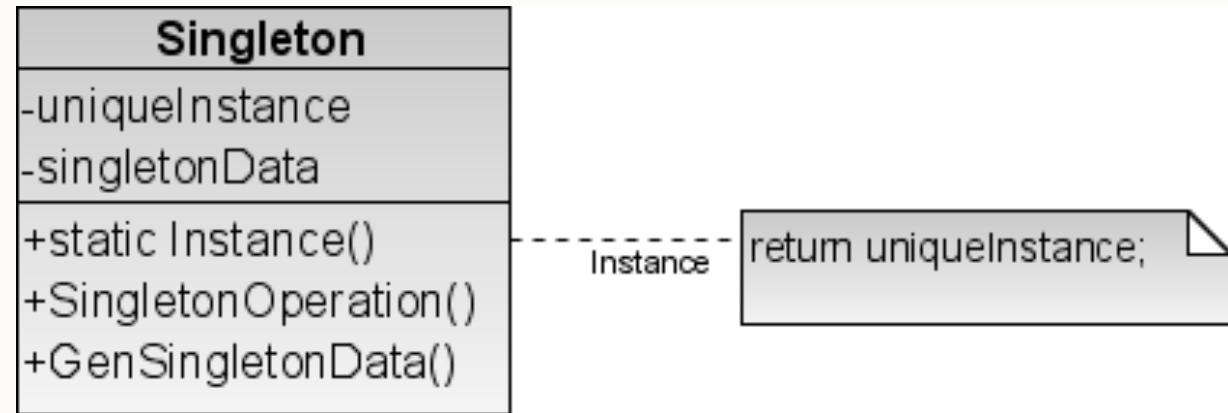
```
public class Singleton {  
    private static Singleton uniqueInstance;  
  
    private Singleton () {}  
  
    public static Singleton getInstance( ) {  
        if ( uniqueInstance == null){  
            uniqueInstance = new Singleton();  
        }  
        return uniqueInstance;  
    }  
}
```

Static variable to hold single instance

Private Constructor

Instantiate class and return a single instance of it

Benefits of the Singleton



- Controlled access to sole instance
- Reduced namespace
- Can easily transform into concrete factory by permitting a variable number of instances



</>

Singleton has some issues though

- There are even more things to consider around how to ensure that there is only a single instance.
 - concurrent requests
 - threading
 - might not have all the information needed at the point of static initialisation
 - interfacing with subclasses



The Factory Patterns

With the Factory Method pattern we create objects without exposing the creation logic to the client and refer to newly created object using a common interface.



About the Factories- The Virtual Constructor

</>

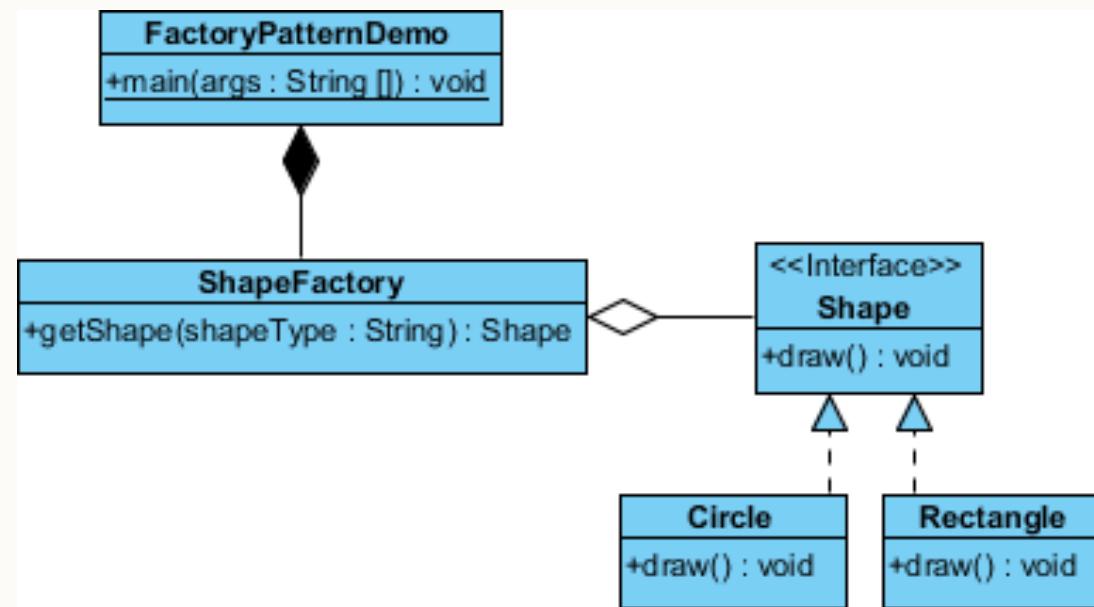
- Creational patterns which abstract the object instantiation process.
- They hide how objects are created and help make the overall system independent of how its objects are created.
- Class creational patterns focus on the use of inheritance to decide the object to be instantiated in the Factory pattern.
- The factory method pattern can save you from those awkward dependencies in OOD by letting you define an interface for creating an object, but let its related subclasses decide which classes to instantiate.
- Used when the class does not know precisely which class of objects it must create.



Factory Method Pattern Example

</>

- We want to create different types of Shape objects





</>

```
1 public class FactoryPatternDemo {  
2  
3 @① public static void main(String[] args) {  
4     ShapeFactory shapeFactory = new ShapeFactory();  
5  
6     //get an object of Circle and call its draw method.  
7     Shape shape1 = shapeFactory.getShape("CIRCLE");  
8  
9     //call draw method of Circle  
10    shape1.draw();  
11  
12    //get an object of Rectangle and call its draw method.  
13    Shape shape2 = shapeFactory.getShape("RECTANGLE");  
14  
15    //call draw method of Rectangle  
16    shape2.draw();  
17  
18 }  
19 }
```

```
1 public interface Shape {  
2     void draw();  
3 }  
4  
1 public class Circle implements Shape {  
2  
3 @② Override  
4     public void draw() {  
5         System.out.println("Inside Circle::draw() method.");  
6     }  
7 }  
8  
1 public class ShapeFactory {  
2  
3     //use getShape method to get object of type shape  
4 @③ public Shape getShape(String shapeType){  
5     if(shapeType == null){  
6         return null;  
7     }  
8     if(shapeType.equalsIgnoreCase("CIRCLE")){  
9         return new Circle();  
10    } else if(shapeType.equalsIgnoreCase("RECTANGLE")){  
11        return new Rectangle();  
12    }  
13  
14    return null;  
15 }  
16  
17 }  
18 }
```



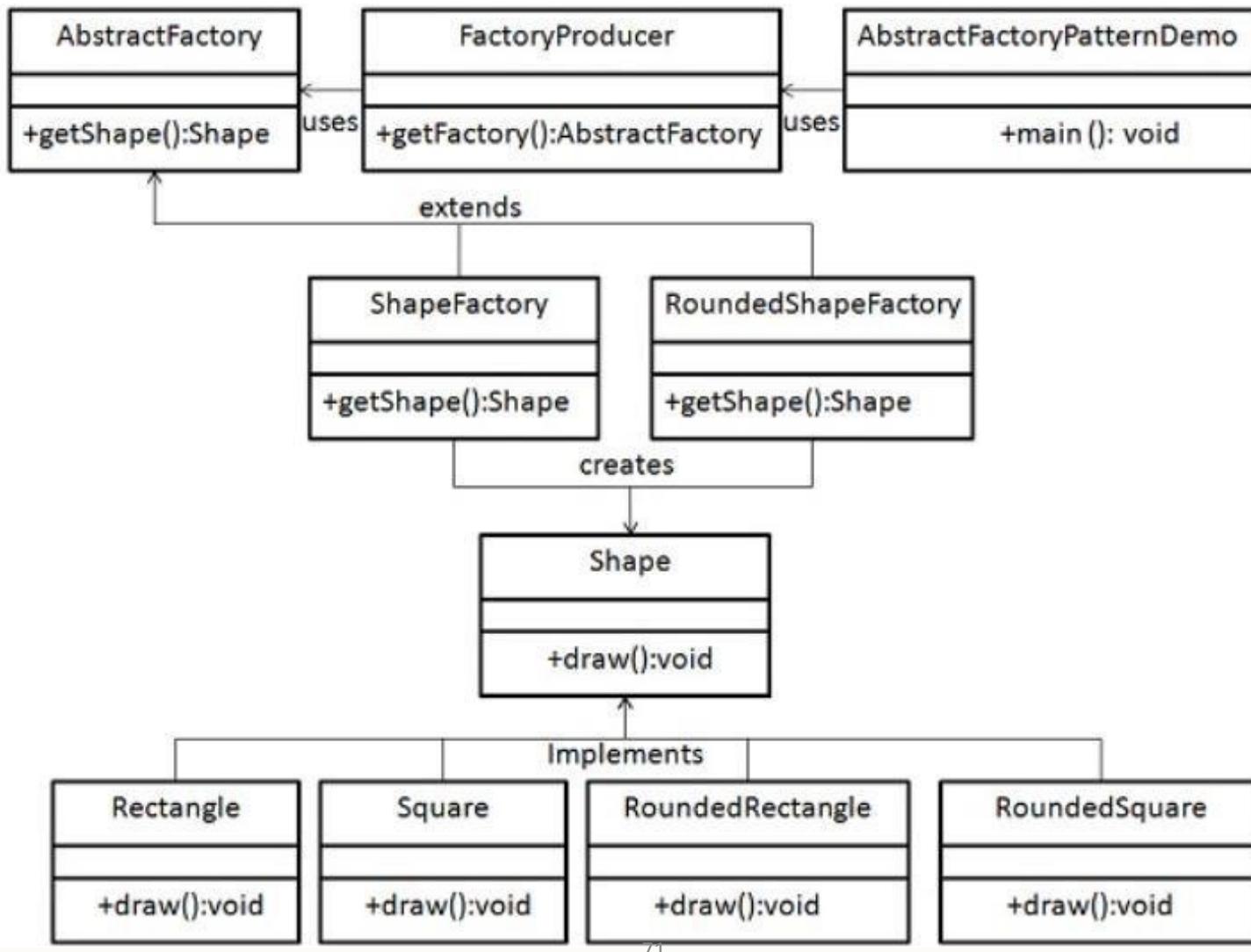
Abstract Factories

**What we just discussed were
concrete factories**

Go one step further with the Abstract Factory

An Abstract Factory is a Factory of Factories

Used when you need an interface for creating
related objects without specifying their classes





Things to note...

- The AbstractFactory class determines the actual type of the concrete object and creates it.
- But it returns an abstract pointer to the concrete object just created.
- This stops the client from knowing anything about the actual creation of the object.
- The objects of the concrete type, created by the factory, are accessed by the client only through the abstract interface.
- To add new concrete types, modify the client code and make it use a different factory.



The Infamous Pizza Factory Example

- The Pizza Factory which defines method names and returns types to make different kinds of pizza.
- The abstract factory can be named **AbstractPizzaFactory**.
- **RomeConcretePizzaFactory** and **MilanConcretePizzaFactory** being two extensions of the abstract class.
- The abstract factory will define types of toppings for pizza, like pepperoni, sausage or anchovy.
- The concrete factories will implement **only a set of the toppings**, which are **specific** for the area.
- More info: <https://www.youtube.com/watch?v=pt1lbV1aSZ4>



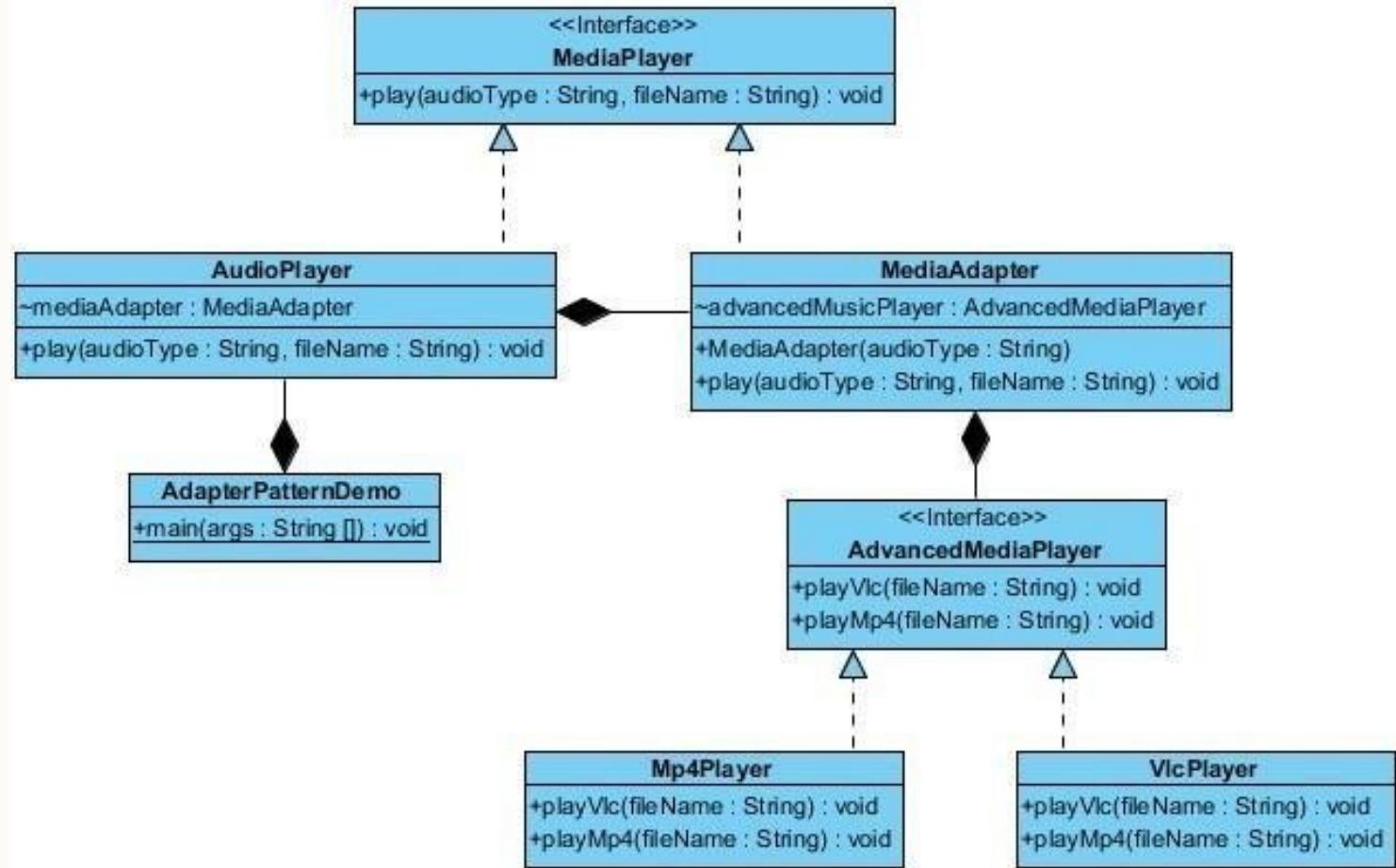
Adapter Pattern (Structural)



- This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces.
 - **Class adapter**: Relies on multiple inheritance
 - **Object adapter**: Relies on object composition >> we use this one!



Adapt Pattern Example



- An audio player device can play mp3 files only and wants to use an advanced audio player capable of playing vlc and mp4 files
- More info: https://www.youtube.com/watch?v=5-xqFjo_jC8



Observer Pattern

- Observer pattern is used when there is one- to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.



Based on the model of old style newspaper subscriptions

- A newspaper publisher goes into business and begins publishing newspapers.
- You subscribe to a particular publisher and every time there is a new edition, it gets delivered to you for as long as you subscribe.
- You unsubscribe when you don't want any more newspapers, delivery also stops.
- While the publisher remains in business, other entities such as other people, hospitals, planes etc subscribe and unsubscribe.





Simple Relationships in the Observer

**Publishers + Subscribers =
Observer**

Publishers = SUBJECT

Subscribers = OBSERVERS



Loose Coupling



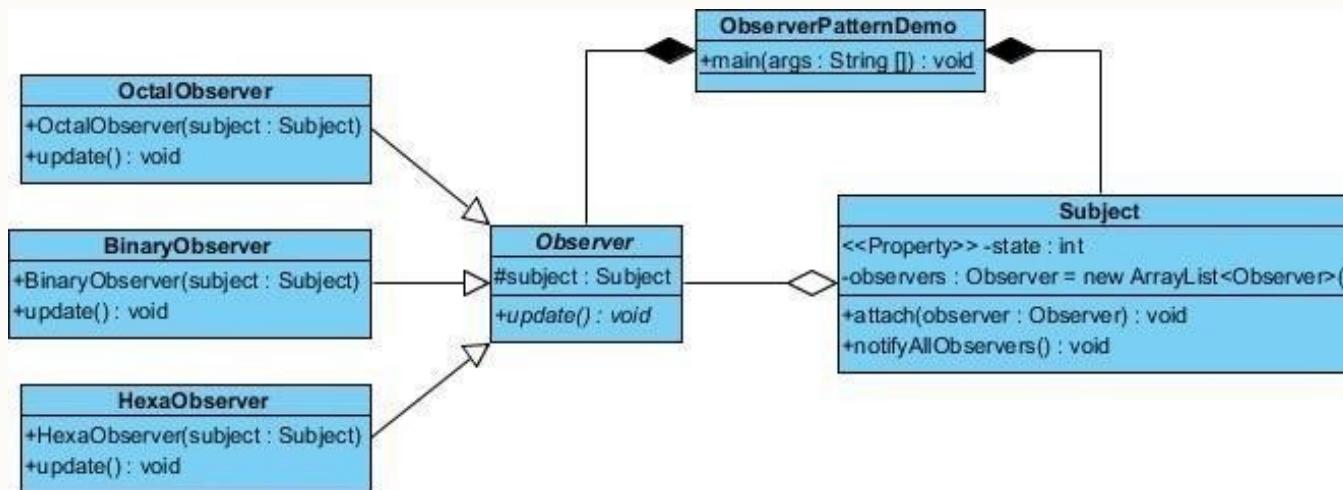
- Two objects are loosely coupled if they interact but have very little knowledge of each other.
- It does not need to know the class of the observer.
- We can add new observers at any time irrespective of the state of the subject object and without modifying the subject object.
- Changes to either subject or object will not affect the other.



Observer Pattern Example

</>

- Number Converter for Programmers





Weather Station Example

◁ ▷

- If we have a IoT box connected to multiple weather stations, how can we set up the observer pattern and make the sensors observable in order to efficiently construct an OO version of this software?
- The weather station will be based on a WeatherData object which tracks current weather conditions including temp, humidity and pressure.
- Create an application which has three display elements:
 - current conditions, weather stats and simple forecast.
- The displays must be updated in real time.
- Have it expandable so that an API can be used to create new displays.



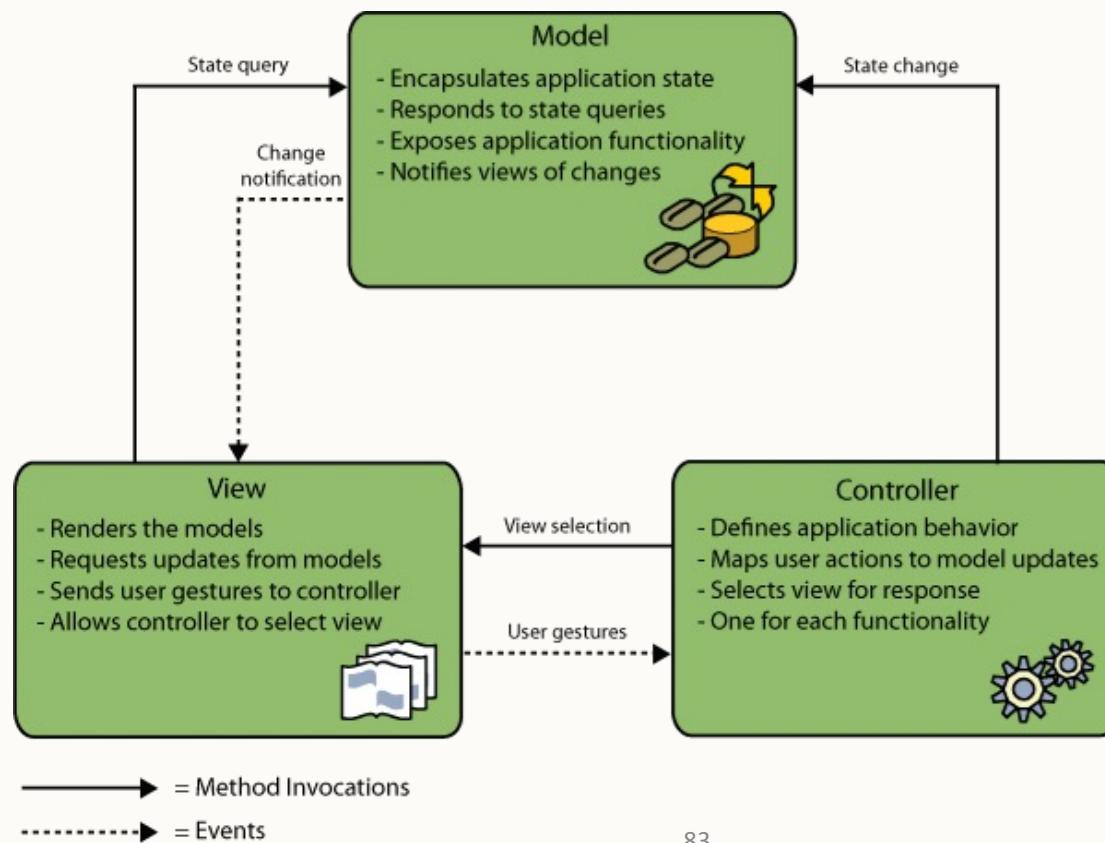
Conveniently Java Loves The Observer

- Inbuilt into the `java.util` and in JDK, in JavaBeans and Swing.
- JButton - `addActionListener` method is the subject's register observer method.
- Smalltalk **Model View Controller** user interface framework.



Oooh and remember MVC?

</>





Good Software Maintenance Avoids 'Antipatterns'

- Experience re-use to avoid common problems in software maintenance.
- Notable examples include:
 - *The Blob*: giving one object the greatest share of the responsibility.
 - *Lava Flow*: where dead or obsolete code is deeply ingrained in the functionality.
 - *Poltergeists*: classes with highly limited roles.
 - *Dead End*: achieved by modifying a reusable component when that component is no longer supported by the vendor.
 - *Spaghetti Code*: code developed in an ad hoc manner which is never refactored.
 - *Cut and Paste programming*: overuse of stackoverflow etc, see last week's every programming tutorial video ever.
- Still confused? Try this: <https://sourcemaking.com/antipatterns/software-development-antipatterns>



Summary



- Recap of the philosophy underlying object orientation.
- SOLID Principles to implement OO in a maintainable manner.
- The standard Design Patterns
 - Overview
 - Singleton
 - Factory(s)
 - Adapter
 - Observer/observable
- Look at practical implementations of design patterns.



Additional Resources

The screenshot shows the tutorialspoint website with a dark blue header. The header includes the tutorialspoint logo, a search bar, and navigation links for HOME, TUTORIALS LIBRARY, CODING GROUND, and SEN. The main content area features a large orange banner with the text "LEARN JAVA DESIGN PATTERNS" and "simply easy learning". Below the banner, there are two smaller images of hands holding books. A navigation menu on the left lists various design patterns: Home, Overview, Factory Pattern, Abstract Factory Pattern, Singleton Pattern, Builder Pattern, Prototype Pattern, Adapter Pattern, and Bridge Pattern. The central content area is titled "Design Patterns in Java Tutorial" and contains a brief introduction to design patterns and a statement about the tutorial's purpose.

http://www.tutorialspoint.com/design_pattern



Useful Resources

The screenshot shows the homepage of SourceMaking.com. At the top right are social media links for Facebook and Twitter, and navigation links for Contact us and Log in. On the left, there's a sidebar with a cartoon illustration of a green, hairy monster sitting at a laptop. The sidebar menu includes links for Premium Stuff, Design Patterns, AntiPatterns, Refactoring, and UML. Below the sidebar is a feedback link: Log in sg/guru:content.feedback. The main content area features a large white robot with its arms raised, with the text "Hello, world!" next to it. A descriptive paragraph explains that the site is about software architecture and provides links for design patterns, anti-patterns, and refactoring. Below this is a section for starting from the very beginning or picking a topic of interest. At the bottom are four smaller images: two pairs of sunglasses with circuit board patterns, a blue control panel with a red button, and a blueprint-style drawing of a complex system.

<https://sourcemaking.com/>



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



Questions?



Lecture 8 – Refactoring

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior



Register your attendance

COMP2013: Developing Maintainable Software
Week 9 – 4:00pm Monday – 20 November 2023



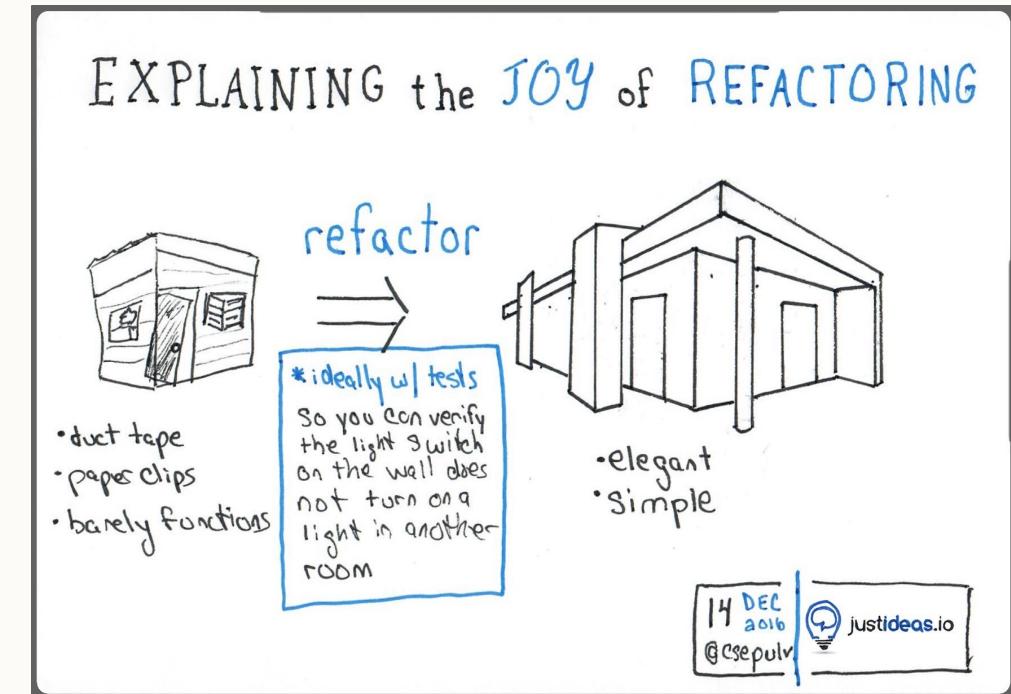
valid for 65 minutes from 3:55pm
generated 2023-10-10 03:14



</>

Today's Learning Objectives

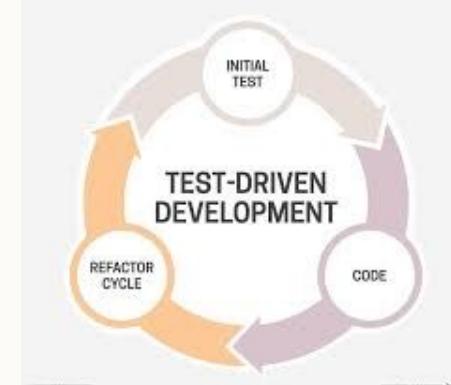
- To be able to assess when code needs refactoring by identifying Code Smells
- To understand the concept of refactoring
 - To refactor small sections of code to increase maintainability
 - To refactor larger code structures for the application of SOLID/Design Patterns
- To understand the links between regression testing, test driven development, refactoring and legacy code



<https://medium.com/justideas-io/explaining-the-joy-of-refactoring-to-the-non-developer-72d97223359c>



The Software Maintenance Starter Pack



rm -rf *



"I've just copied my code from stack overflow...its bound to work"

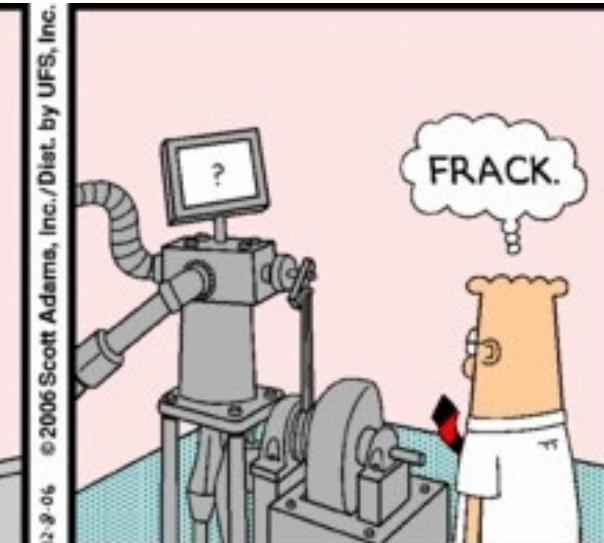
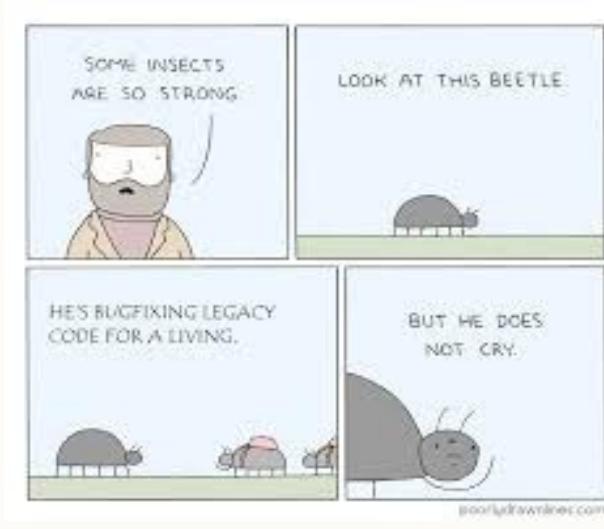


"Can I exceed the word limit on CWK1?"



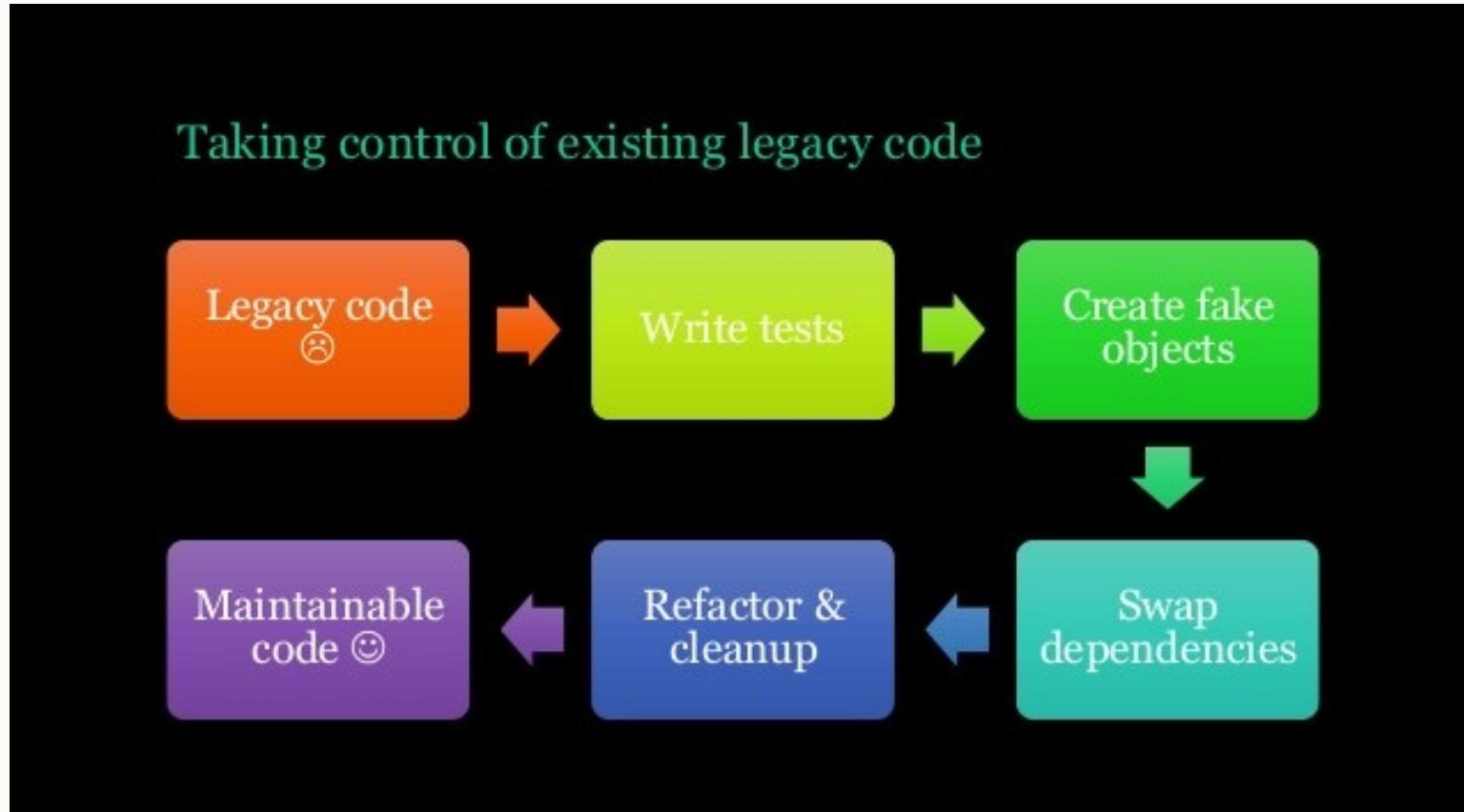


Legacy Code Maintenance The Core Of An SE Role





The Process of Working With Legacy Code



Still confused? Try this: <https://www.slideshare.net/dhelper/working-with-c-legacy-code>



Reality Reality check!

A quick reality check

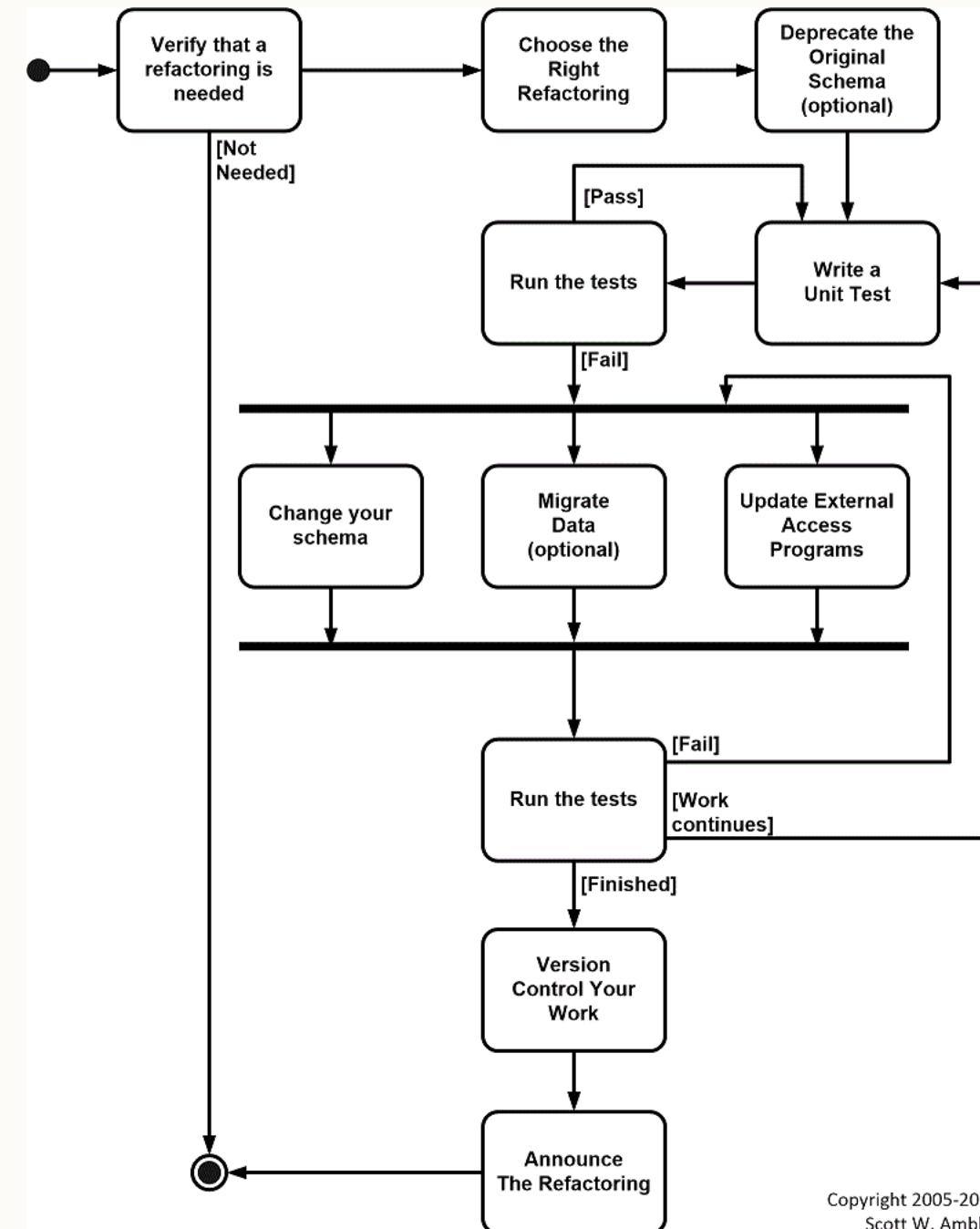
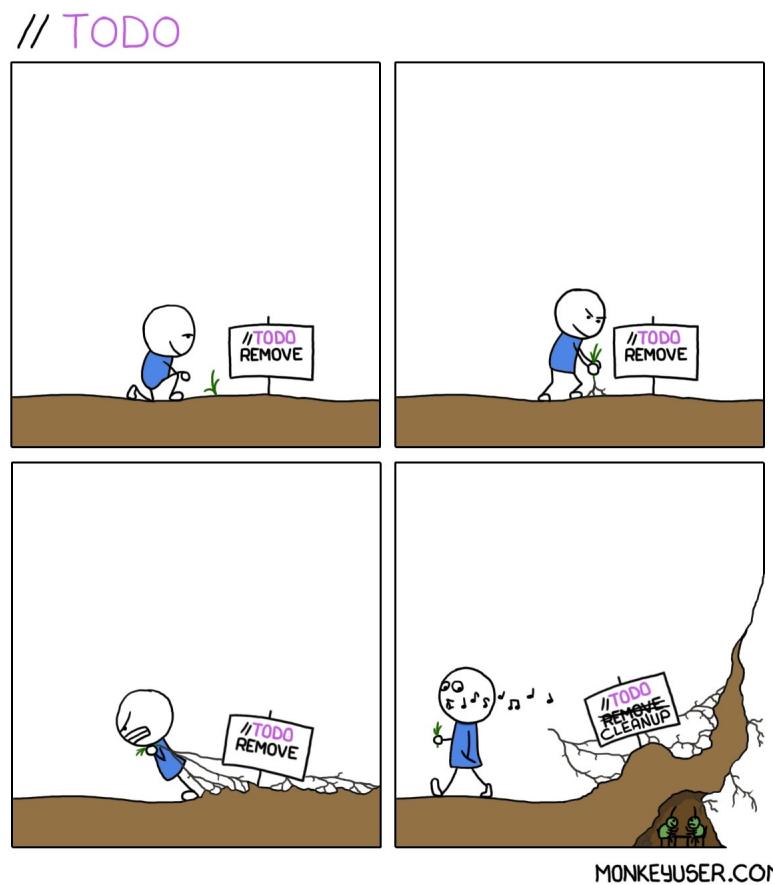
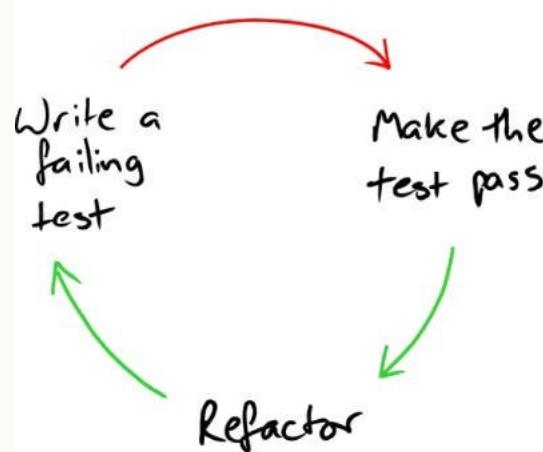
- Code have dependencies
- Refactoring == code change
- Code change == breaking functionality (78.3%)
- Breaking functionality → go home == false

Source : <https://www.slideshare.net/dhelper/working-with-c-legacy-code>



Refactoring

Easy in theory





Quotes about refactoring from the authorities

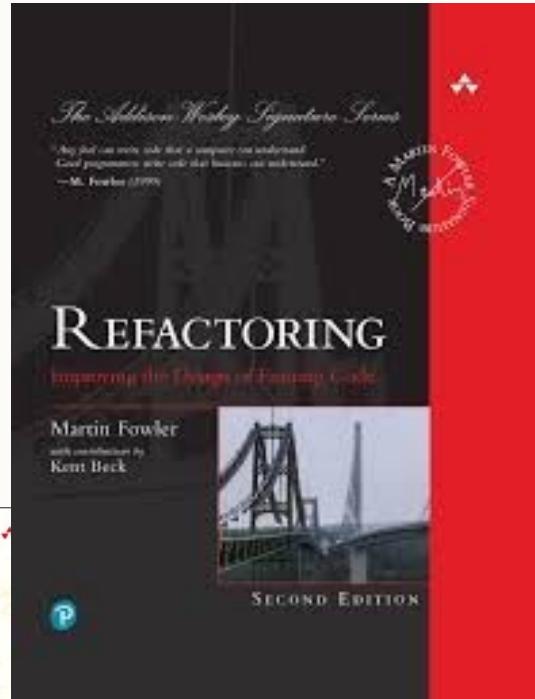
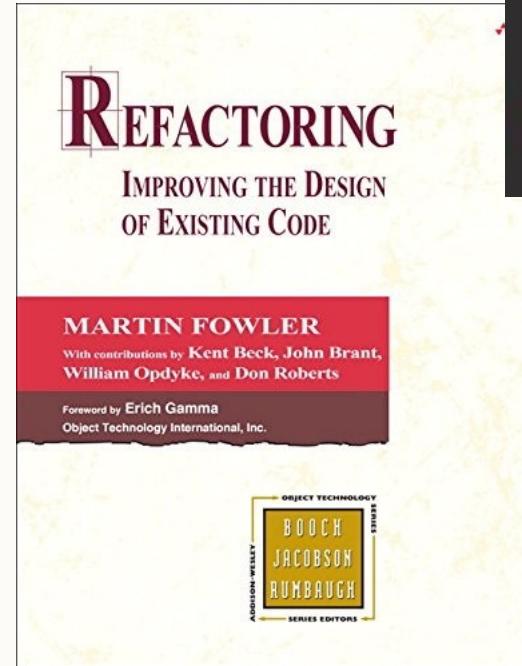
- *“A series of small decisions and actions all made through the filter of a set of values and the desire to make something excellent”* – Chad Fowler, Ruby Central
- *“Refactoring is the process of changing a software system in a way that does not alter the external behavior of the code yet improves its internal structure”* – Martin Fowler
- *“The XP philosophy is to start where you are now and move towards the ideal. From where you are now, could you improve a little bit?”* - Kent Beck
- *“Any fool can write code that a computer can understand. Good programmers can write code that humans can understand”* – Martin Fowler



Kent Beck
Agile Guru &
Code Smeller



Martin Fowler
Refactoring Godfather





Refactoring improves existing code

- The process of changing a software system to not alter the external behavior of the code yet improves its internal structure in an incremental fashion
- A disciplined way to clean up code to minimize the chance of introducing bugs
- Improving the design of code after it has been written
 - Can be code you wrote yesterday or code written decades ago
- Explores the tradeoff space between clarity and performance
- Refactoring is not optimizing code to make it run faster
 - It is about making code make more sense and increasing its robustness
 - Making code open for modification
 - Application of SOLID principles and design pattern



Kent Beck Says...

“The majority of the cost of software is incurred after the software has been first deployed. Thinking about my experience of modifying code, I see that I spend much more time reading the existing code than I do writing new code. If I want to make my code cheap, therefore, I should make it easy to read.”

```
/**  
 * Code Readability  
 */  
if (readable()) {  
    be_happy();  
} else {  
    refactor();  
}
```



A Trivial HTML Example Of Using Whitespace in Code

```
<html> <head> <title>Make your HTML code readable by using white space</title> </head> <body> <p>Do you want an easy way to understand your code? If yes, use white space to organize your code. You can use white space as much as possible to make your code readable. You will be surprised how easy it is to follow your code when you use appropriate amount of white space. If you do not use white space, your HTML code will look messy. So: </p> <ul> <li>group related tags and separate them by indenting</li> <li>use white space</li> <li>use comments when necessary to explain what your code does</li> </ul> </body> </html>
```

```
<html>
  <head>
    <title>
      Make your HTML code readable by using white space
    </title>
  </head>
  <body>
    <p>
      Do you want an easy way to understand your code? If yes, use white space to organize your code. You can use white space as much as possible to make your code readable. You will be surprised how easy it is to follow your code when you use appropriate amount of white space. If you do not use white space, your HTML code will look messy. So:
    </p>
    <ul>
      <li>group related tags and separate them by indenting</li>
      <li>use white space</li>
      <li>use comments when necessary to explain what your code does</li>
    </ul>
  </body>
</html>
```



Who is involved in Refactoring?

- Refactoring is a multidisciplinary activity involving:
 - Programmers/Developers
 - Senior designers
 - Management
 - System Architects
- Each of these roles will adapt the principles of refactoring to a specific project or workspace
- Refactoring occurs in different forms across a project



Refactoring Roadmap

Quick Wins

- Remove dead code
- Remove code duplicates
- Enhance identifier naming
- Reduce method size

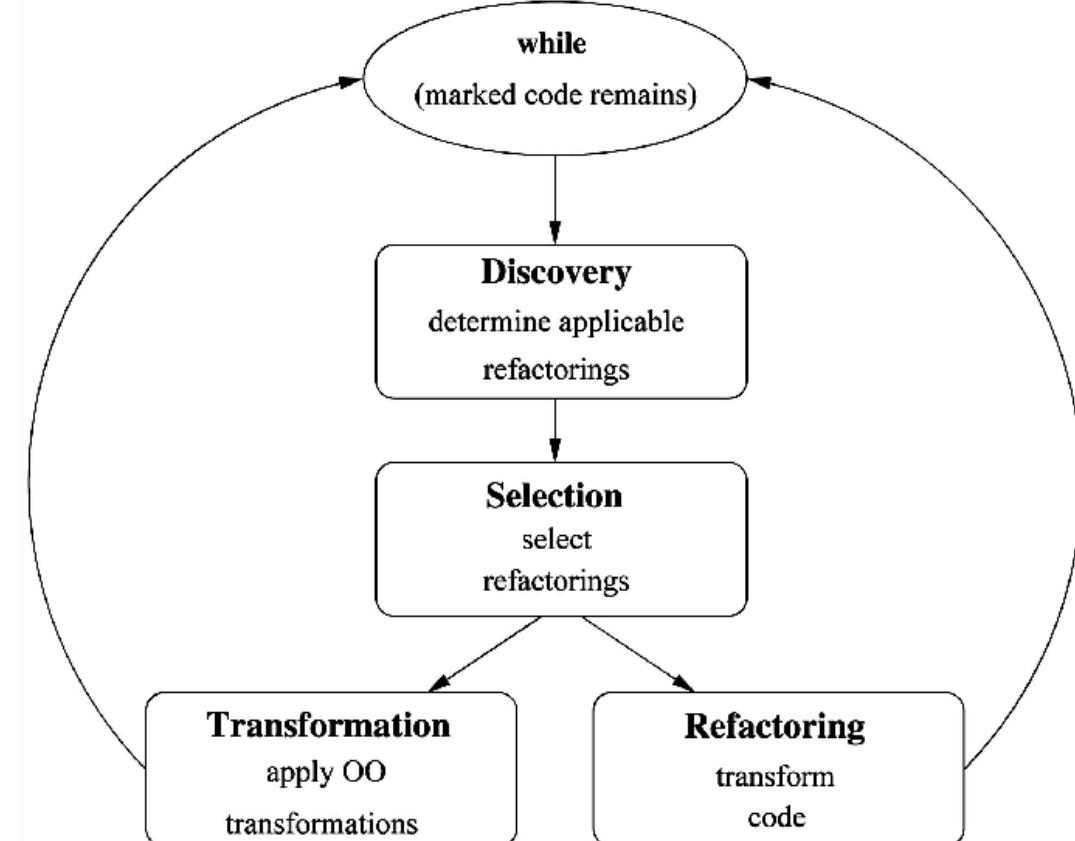
Divide & Conquer

- Discover & split code into components
- Enhance component encapsulation
- Reduce coupling

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Continuous Review





Build Your Test Scaffold Before You Touch Anything

- The key to successful refactoring is to **AVOID** breaking anything while you make your changes
- This requires that you use the existing or develop a new test scaffold
 - Regression testing, right?
- Run the test scaffold after every instance of change, not after every feature refactored
- Legacy code may not include any test cases, in which case you need to add them before you start ‘tinkering’ i.e. making incremental changes
- Substitute ‘live’ components with dummy ones i.e. use ‘Mock Objects’
- Once you have done this you are ready to go
- Still Confused? Try this:

<https://martinfowler.com/articles/mocksArentStubs.html>



Identify Code Smells

Decide what you are going to refactor



Code Smells Because Its Hard to Understand (not because it is incorrect)

- Duplicated Code:
 - bad because if you modify one instance of duplicated code but not the others, you (may) have introduced a bug!
- Long Methods:
 - long methods are more difficult to understand
 - Note: performance concerns with respect to lots of short methods are largely unfounded
- Large Classes:
 - classes try to do too much, which reduces cohesion (violates single responsibility, “The Blob”)
- Long Parameter Lists:
 - hard to understand, can become inconsistent
- Divergent Changes:
 - related to cohesion where one type of change requires changing one subset of methods; another type of change requires changing another subset



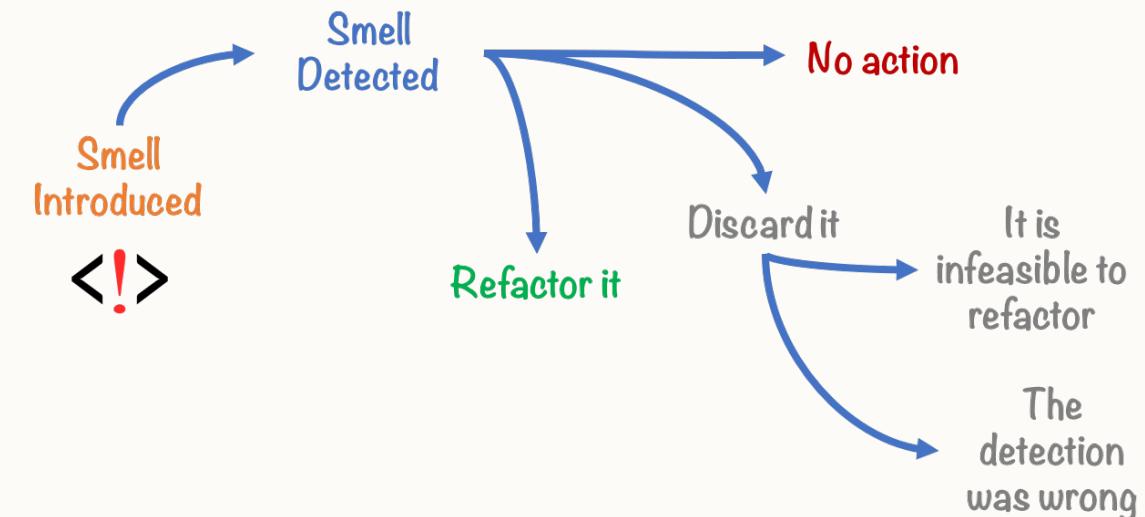
More code smells examples

- Lazy Classes:
 - A class that no longer “pays its way”, part formed functionality
- Speculative Generality:
 - “Oh I think we need the ability to do this kind of thing someday”
- Temporary Field:
 - An attribute of an object is only set in certain circumstances; but an object should need all of its attributes
- Data Class
 - These are classes that have fields, getting and setting methods for the fields, and nothing else; they are data holders, but objects should be about data AND behavior
- Refused Bequest
 - A subclass ignores most of the functionality provided by its superclass where the subclass may not pass the “IS-A” test (Liskov Substitution Violation)
- Comment-based coverup
 - Comments are sometimes used to hide bad code, symptomatic of large blocks of comments, not using automatic comment generation



Still Confused? Try This:

- Many more smells at:
<http://c2.com/cgi/wiki?CodeSmell>
- Matching up a smell to its refactoring counterpart:
<http://wiki.java.net/bin/view/People/SmellsTORefactorings>
- Taxonomy of code smells:
<http://www.tusharma.in/smells/>
- An easy read:
<https://medium.com/@tusharma/how-to-track-code-smells-effectively-48dbf5ba659d>





Refactoring Code

Shamelessly Inspired By Refactoring (Fowler), Chapters 6-10



Fowler Defined Different Categories of Refactoring

- *Composing Methods*: refactoring within a method or within an existing class
- *Moving Features Between Objects*: refactoring changing the responsibility of a class
- *Organizing Data*: refactoring to improve data structures and object linking
- *Simplifying Conditional Expressions*: encapsulation of conditions by replacing with polymorphism
- *Making Method Calls Simpler*: refactorings that make interfaces simpler
- *Dealing with Generalization*: moving methods up and down the hierarchy of inheritance by (often) applying the Factory or Template design pattern
- *Big Refactorings*: architectural refactoring to promote loose coupling or to realise a redesign via object orientation (Note: this is extremely difficult for most projects).



Refactoring reduces repetition and helps focus on objects

- Adding a new animal type, such as **Amphibian**, does not require revising and recompiling existing code
 - We have provided good extensibility which is less likely to break future code
- Mammals, birds, and reptiles are likely to differ in other ways, and we've already separated them out
 - Therefore we won't need more switchstatements in future
- Eliminated the 'flags' we needed to tell one kind of animal from another i.e. elegantly removed MAMMAL=0 etc.
- We now use Objects the way they were meant to be used and give our code better structure

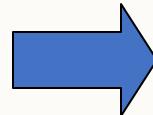


Encapsulate Fields to retain private variables

- Un-encapsulated data is a violation of key object-oriented principles
- Use property get and set procedures to provide public access to private (encapsulated) member variables

```
public class Course
{
    public List students;
}
```

```
int classSize =
course.students.size();
```



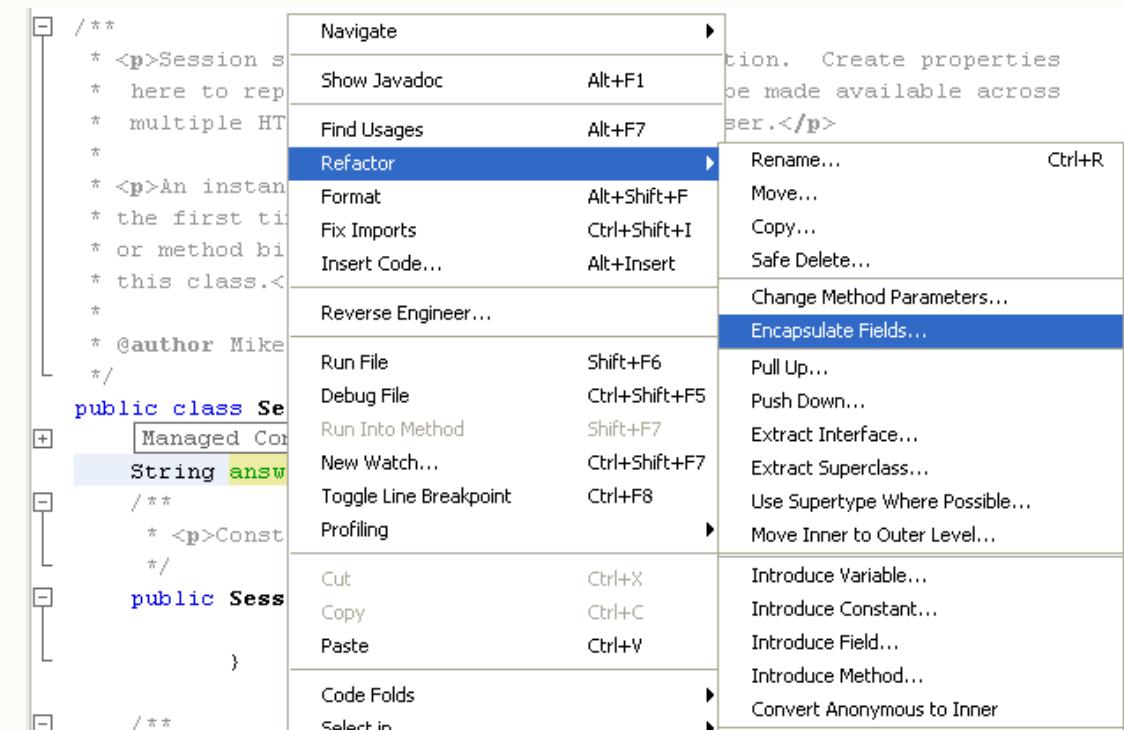
```
public class Course
{
    private List students;
    public List getStudents()
    {
        return students;
    }
    public void setStudents(List s)
    {
        students = s;
    }
}
```

```
int classSize = course.getStudents().size();
```



Encapsulating Fields Automatically With IDEs

- I have a class with 10 fields, its quite an overhead to manage and is a pain to have to go through to add `get_()` and `set_()` for each one
- Refactoring Tools
 - See NetBeans/Eclipse/Visual Studio/ Monodevelop's refactoring tools
- Also allows for automating two other refactoring tasks
 - Rename Method: cascading the new name of the method
 - Change Method Parameters: auto update when parameters are modified





Extract method from a larger block of code

- If a method is too long or needs excessive code to describe its function, then you can probably extract a new method out of the code
- Short, well named methods are easier to maintain
- Obeys the single responsibility principle
- How big is too big?
 - Used to be around 125 LOC in Fortran
 - Then we have 80 lines of code (from the size of the UNIX terminal)
- Length is not the issue – its down to “the semantic distance between a method name and the method body



Create a new method to perform the extract method

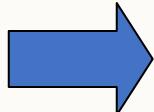
1. Create a new method and name it after the *intention* of the method
 - If you can't derive this name, you probably should not be extracting the method!
2. Copy the extracted code from the source method into the new method
3. Scan the extracted code for references to local and temporary variables
4. Check if any of the local variables are modified by the extracted code
 - See if this can be transformed into a query instead using the Replace Temp with Query refactor (more on this later)
5. Pass into new method as parameters any required local scope variables
6. Replace the extracted code with a method call to the new method, checking the use of temp vars
7. Run your regression tests



Using Extract Method is a quick win!

- What are the code smells? See how the comment becomes the method name...

```
public class Customer
{
    void int foo()
    {
        ...
        // Compute score
        score = a*b+c;
        score *= xfactor;
    }
}
```



```
public class Customer
{
    void int foo()
    {
        ...
        score = ComputeScore(a,b,c,xfactor);
    }

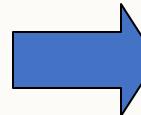
    int ComputeScore(int a, int b, int c, int x)
    {
        return (a*b+c)*x;
    }
}
```



Replace Parameter with Explicit Method

- You have a method that runs different code depending on the values of an enumerated parameter. Create a separate method for each value of the parameter.

```
void setValue (String name, int value) {  
    if (name.equals("height")) {  
        height = value;  
        return;  
    }  
    if (name.equals("width")) {  
        width = value;  
        return;  
    }  
    Assert.shouldNeverReachHere();  
}
```



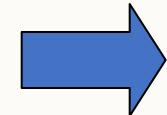
```
void setHeight(int arg)  
{  
    height = arg;  
}  
  
void setWidth (int arg)  
{  
    width = arg;  
}
```



Inline Method to reduce indirection

- The method does in the body precisely the method intention

```
class PizzaDelivery
{
    // ...
    int getRating() {
        return moreThanFiveLateDeliveries() ? 2 : 1;
    }
    boolean moreThanFiveLateDeliveries() {
        return numberOfLateDeliveries > 5;
    }
}
```



```
class PizzaDelivery
{
    // ...
    int getRating() {
        return numberOfLateDeliveries
            > 5 ? 2 : 1;
    }
}
```



Replace Temp with Query

- You place the result of an expression in a local variable for later use in your code??
- Move the entire expression to a separate method and return the result from it.
- Query the method instead of using a variable. Incorporate the new method in other methods, if necessary.

```
double calculateTotal() {  
    double basePrice = quantity * itemPrice;  
    if (basePrice > 1000) {  
        return basePrice * 0.95;  
    }  
    else  
        return basePrice * 0.98;  
}
```

```
double calculateTotal() {  
    if (basePrice() > 1000) {  
        return basePrice() * 0.95; }  
    else {  
        return basePrice() * 0.98; }  
}  
double basePrice() {  
    return quantity * itemPrice;  
}
```

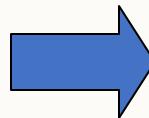




Rename Variable or Method To Self Document

- Perhaps one of the simplest, but one of the most useful that bears repeating: If the name of a method or variable does not reveal its purpose then change the name of the method or variable.

```
public class Customer
{
    public double getinvcdtlmt();
}
```



```
public class Customer
{
    public double getInvoiceCreditLimit();
}
```



Remove assignments to parameters

- Some value is assigned to a parameter inside method's body, therefore use a local variable instead of a parameter
- If a parameter is passed by reference, then after the parameter value is changed inside the method, this value is passed to the argument that requested calling this methods
 - This occurs accidentally and leads to unfortunate effects
 - Even if parameters are usually passed by value (and not by reference) in your programming language, this coding quirk may alienate those who are unaccustomed to it.
- Also, multiple assignments of different values to a single parameter make it difficult for you to know what data should be contained in the parameter at any particular point in time, makes testing harder too



Remove assignments to parameters

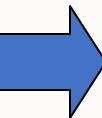
- Each element of the program should be responsible for only one thing. This makes code maintenance much easier going forward, since you can safely replace code without any side effects
- This refactoring helps to extract repetitive code to separate methods



Remove assignments to parameters

- Create a local variable and assign the initial value of your parameter
- In all method code that follows this line, replace the parameter with your new local variable

```
int discount (int inputVal, int quantity)
{
    if (inputVal > 50)
    {
        inputVal -= 2;
    }
    // ...
}
```



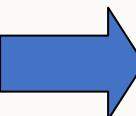
```
int discount (int inputVal, int quantity)
{
    int result = inputVal;
    if (inputVal > 50)
    {
        result -= 2;
    }
    // ...
}
```



Replace Magic Number with Symbolic Content

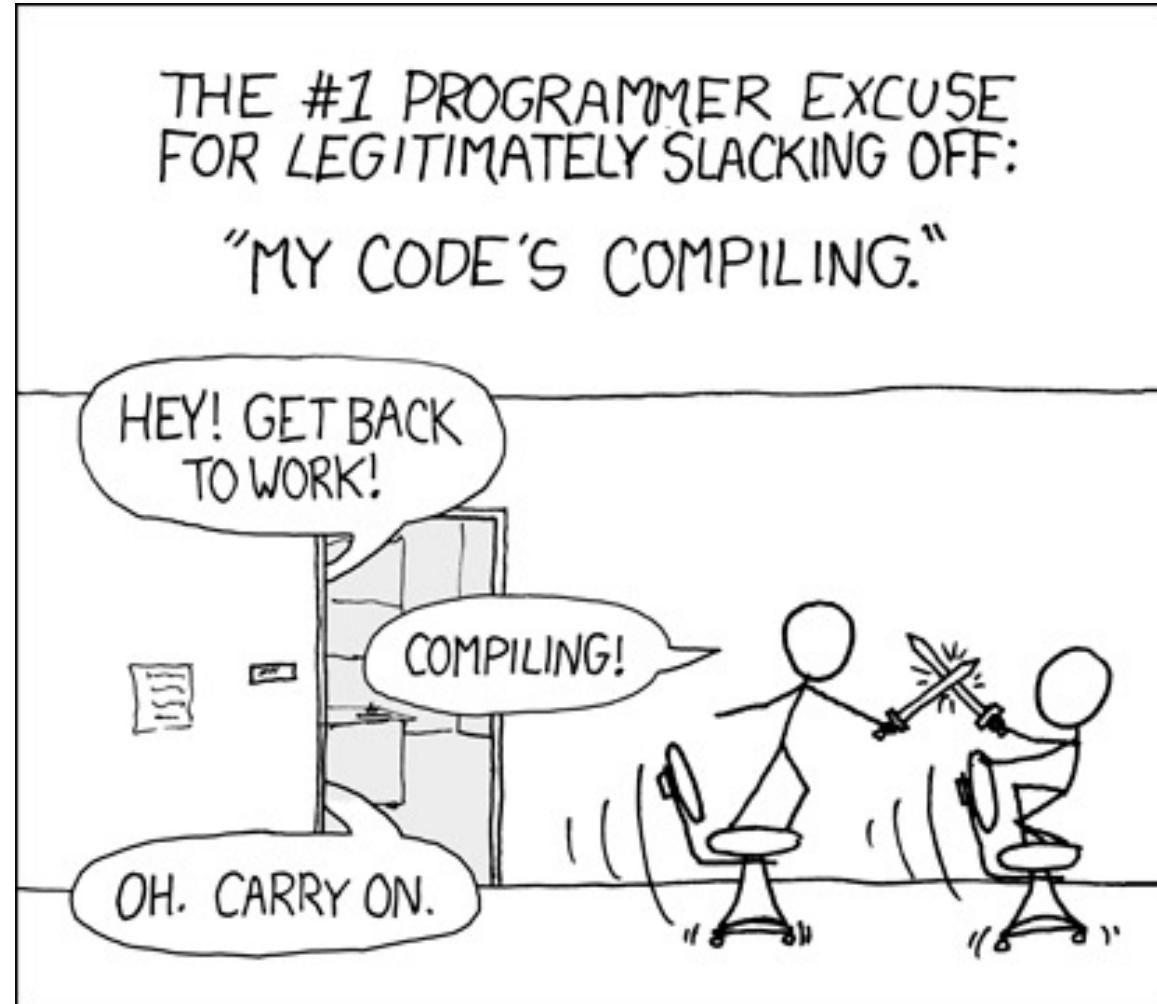
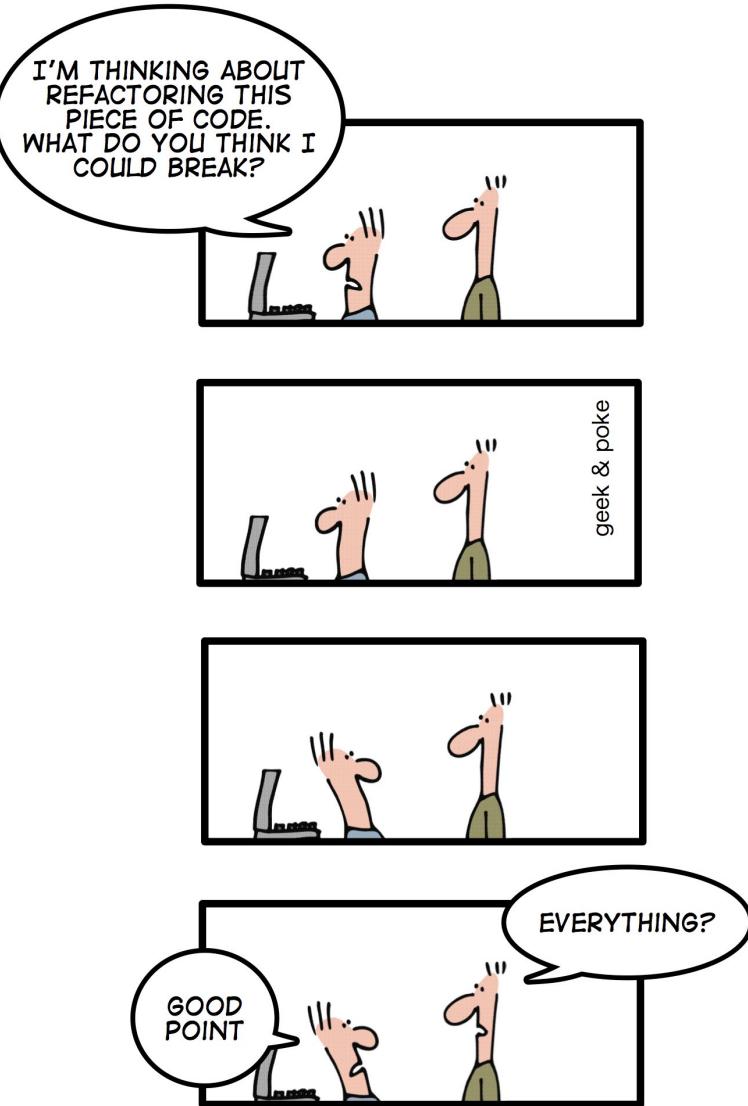
- Used if you have a literal number with a particular meaning
- Create a constant, name it after the meaning and replace the number with it
- This works if this is a universal constant in your code! (global variables are bad)

```
double energy (double mass, double height)
{
    return mass*height*9.81;
// ...
}
```



```
static final double GRAVITY = 9.81;
double potentialEnergy (double mass,
double height) {
    // ...
    return mass * height * GRAVITY;
}
```

Still confused? Try this: <https://www.youtube.com/watch?v=owFFVQYW1p8>





Refactoring at a higher Level

Designing classes and really refactoring to be clean and object oriented



Replace type code with polymorphism

- **Switch statements are very rare in properly designed object-oriented code, much more common in procedural programs**
 - Therefore, a switch statement is a simple and easily detected “bad smell”
 - Of course, not all uses of switch are bad
 - A switch statement should *not* be used to distinguish between various kinds of object
 - Lengthy to test all of the test cases
- There are several well-defined refactorings for this case
 - The simplest is the creation of subclasses



Replace type code with polymorphism

```
class Animal {  
    final int MAMMAL = 0, BIRD = 1, REPTILE = 2;  
    int myKind; // set in constructor  
    ...  
    String getSkin() {  
        switch (myKind) {  
            case MAMMAL: return "hair";  
            case BIRD: return "feathers";  
            case REPTILE: return "scales";  
            default: return "skin";  
        }  
    }  
}
```



Improvement over switch using `extends`

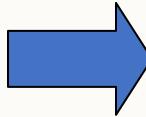
```
class Animal {  
    String getSkin() { return "skin"; }  
}  
class Mammal extends Animal {  
    String getSkin() { return "hair"; }  
}  
class Bird extends Animal {  
    String getSkin() { return "feathers"; }  
}  
class Reptile extends Animal {  
    String getSkin() { return "scales"; }  
}
```



Single Responsibility with Extract Class

- When one class does the job of two, it promotes tight coupling
- Break one class into two, e.g. Having the phone details as part of the Customer class is not a realistic OO model, and also breaks the Single Responsibility design principle. We can refactor this into two separate classes, each with the appropriate responsibility.

```
public class Customer
{
    private String name;
    private String workPhoneAreaCode;
    private String workPhoneNumber;
}
```



```
public class Customer
{
    private String name;
    Private Phone workPhone;
}

public class Phone
{
    private String areaCode;
    private String number;
}
```



Generalization with Extract Interface

- Extract an interface from a class. Some clients may need to know a Customer's name, while others may only need to know that certain objects can be serialized to XML. Having `toXml()` as part of the Customer interface breaks the Interface Segregation design principle which tells us that it's better to have more specialized interfaces than to have one multi-purpose interface.
- How to do this:
 1. Create an empty interface.
 2. Declare common operations in the interface.
 3. Declare the necessary classes as implementing the interface.
 4. Change type declarations in the client code to use the new interface.



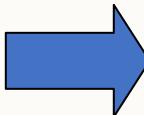
Extract Interface

```
public class Customer
{
    private String name;

    public String getName() { return name; }

    public void setName(String string)
    { name = string; }

    public String toXML()
    { return "<Customer><Name>" +
        name + "</Name></Customer>";
    }
}
```



```
public class Customer implements SerXML
{
    private String name;

    public String getName() { return name; }

    public void setName(String string)
    { name = string; }

    public String toXML()
    { return "<Customer><Name>" +
        name + "</Name></Customer>";
    }
}
```

```
public interface SerXml {
    public abstract String toXML();
}
```



Push Down/Pull Up of methods

- Eliminating duplicate behavior is a core value of refactoring
 - “breeding ground” for bugs
 - Whenever there is duplication there is the risk that you will alter one instance and not the other one
- ‘Pull Up’ when you have methods of the same composition
 - Often after you have refactored a bit to reduce the temp variables
 - Use when you have a subclass which overrides a superclass method yet does the same thing or after applying Substitute Algorithm to make them identical
 - If you have two methods which are similar but not identical use ‘Form Template’ method
- Also applies to the refactoring of a module hierarchy where a method is duplicated on two or more classes, moving the method up the hierarchy
- Push down is the opposite, to specialize a class to stop a method implemented in a superclass which is only used by one or a small number of subclasses



Extras: Move Method - Before

- If a method on one class uses (or is used by) another class more than the class on which its defined, move it to the other class

```
public class Student
{
    public boolean isTaking(Course course)
    {
        return (course.getStudents().contains(this));
    }
}

public class Course
{
    private List students; public
    List getStudents()
    {
        return students;
    }
}
```



Extras: Move Method - Refactored

- The student class now no longer needs to know about the Course interface, and the isTaking() method is closer to the data on which it relies - making the design of Course more cohesive and the overall design more loosely coupled

```
public class Student
{
}

public class Course
{
    private List students;
    public boolean isTaking(Student student)
    {
        return students.contains(student);
    }
}
```

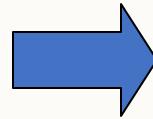


Extras: Introduce Null Object

- If relying on null for default behavior, use inheritance instead

```
public class User
{
    Plan getPlan()
    {
        return plan;
    }
}
```

```
if (user == null)
    plan = Plan.basic();
else
    plan = user.getPlan();
```



```
public class User
{
    Plan getPlan()
    {
        return plan;
    }
}

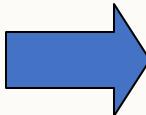
public class NullUser extends User
{
    Plan getPlan()
    {
        return Plan.basic();
    }
}
```



Extras: Replace Error Code with Exception

- A method returns a special code to indicate an error is better accomplished with an Exception.

```
int withdraw(int amount)
{
    if (amount > balance)
        return -1;
    else {
        balance -= amount;
        return 0;
    }
}
```

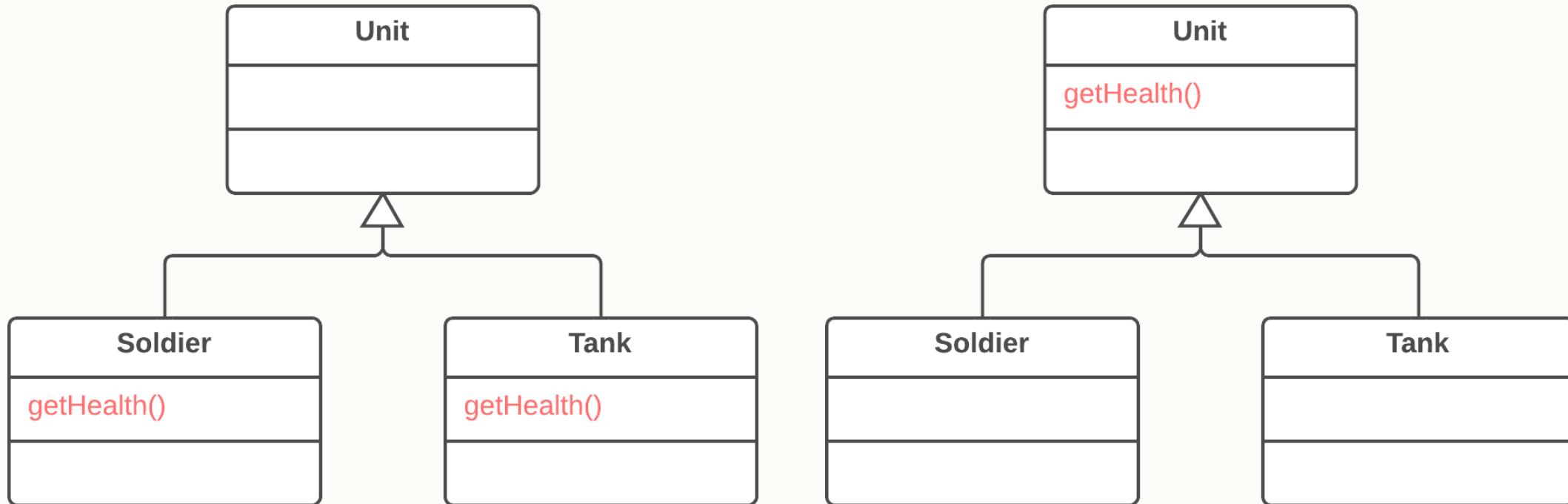


```
void withdraw(int amount)
    throws BalanceException
{
    if (amount > balance)
    {
        throw new BalanceException();
    }
    balance -= amount;
}
```



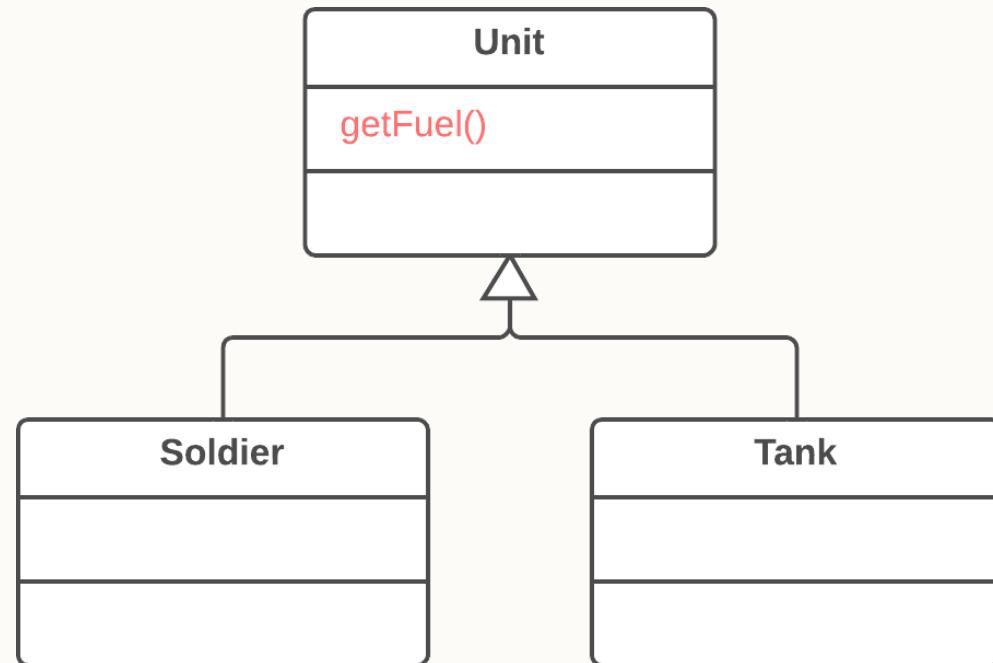
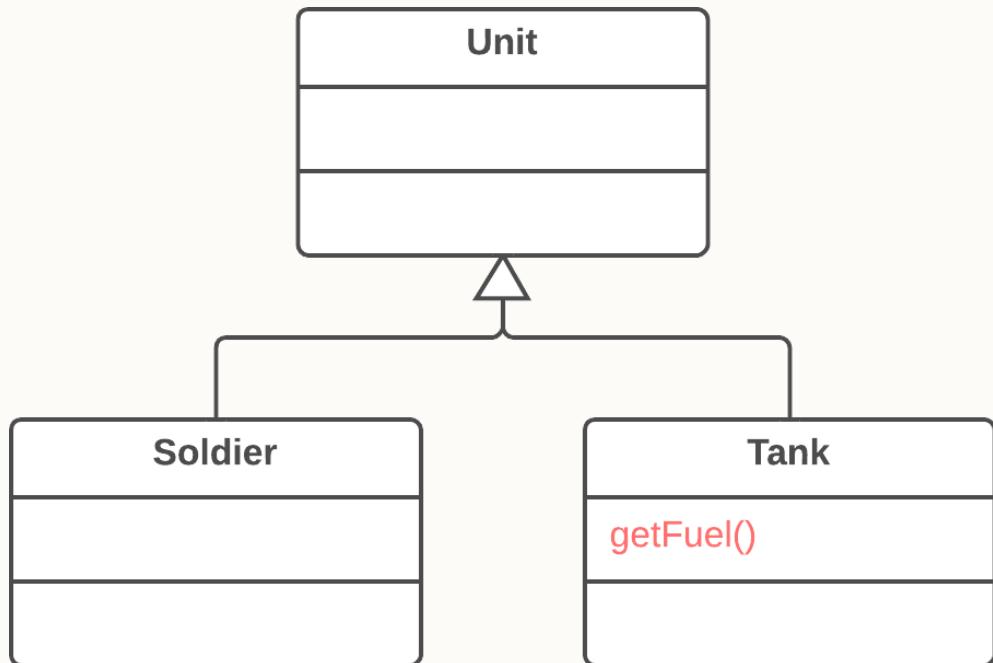
Pull Up

</>





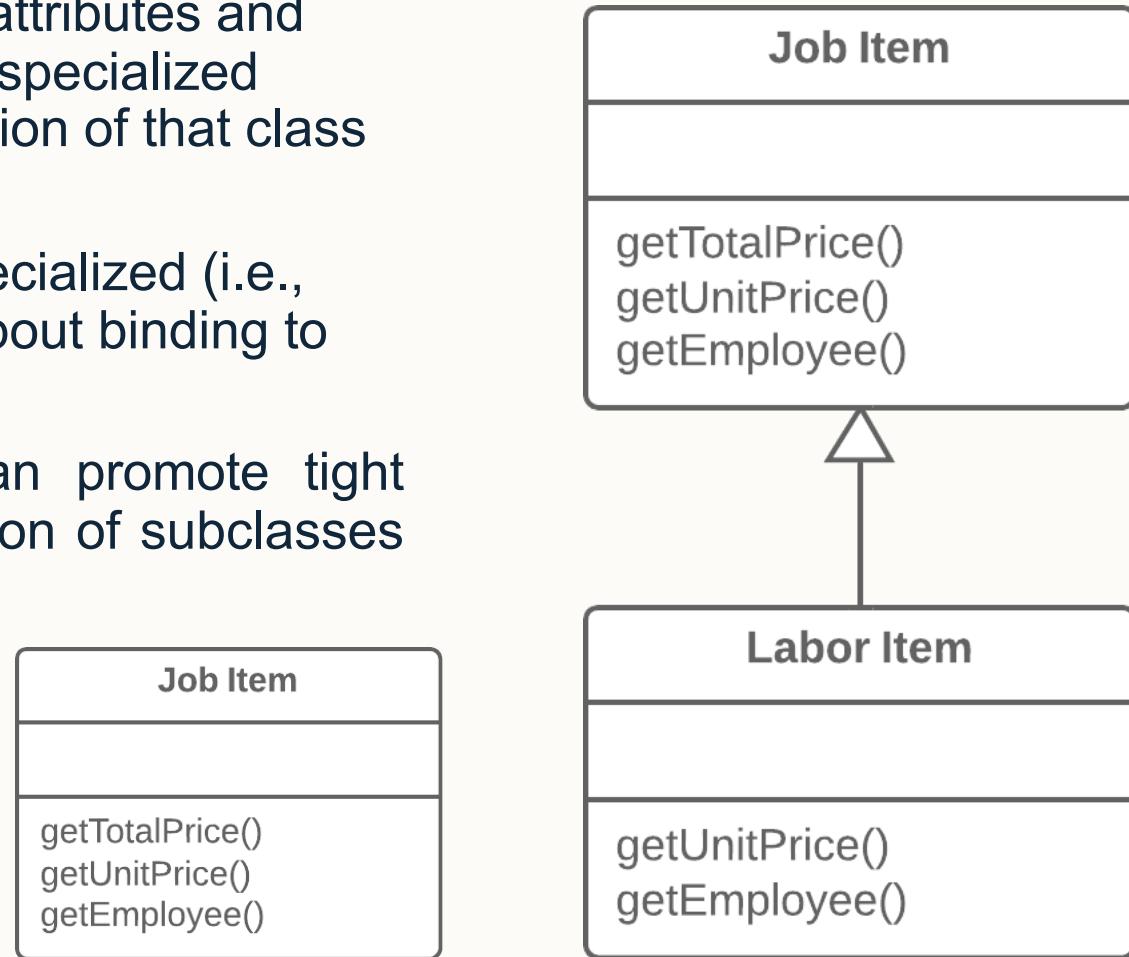
Push Down





You can inherit if you need to with Extract Subclass

- You have found a class has features (attributes and methods) that would only be useful in specialized instances, we can create a specialization of that class and give it those features
- This makes the original class less specialized (i.e., more abstract), and good design is about binding to abstractions wherever possible
- Caution – too much inheritance can promote tight coupling, like making a large collection of subclasses you may want to avoid

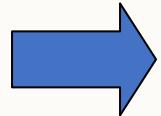




Extract subclass example

</>

```
public class Person
{
    private String name;
    private String jobTitle;
}
```



```
public class Person
{
    protected String name;
}

public class Employee extends Person
{
    private String jobTitle;
}
```

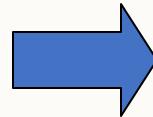


You may also want to Extract Super Classes

- When you find two or more classes that share common features, consider abstracting those shared features into a super-class
- This makes it easier to produce an abstraction, and removes duplicate code from the original classes
- Not used if superclass already exists

```
public class Employee
{
    private String name;
    private String jobTitle;
}

public class Student
{
    private String name;
    private Course course;
}
```



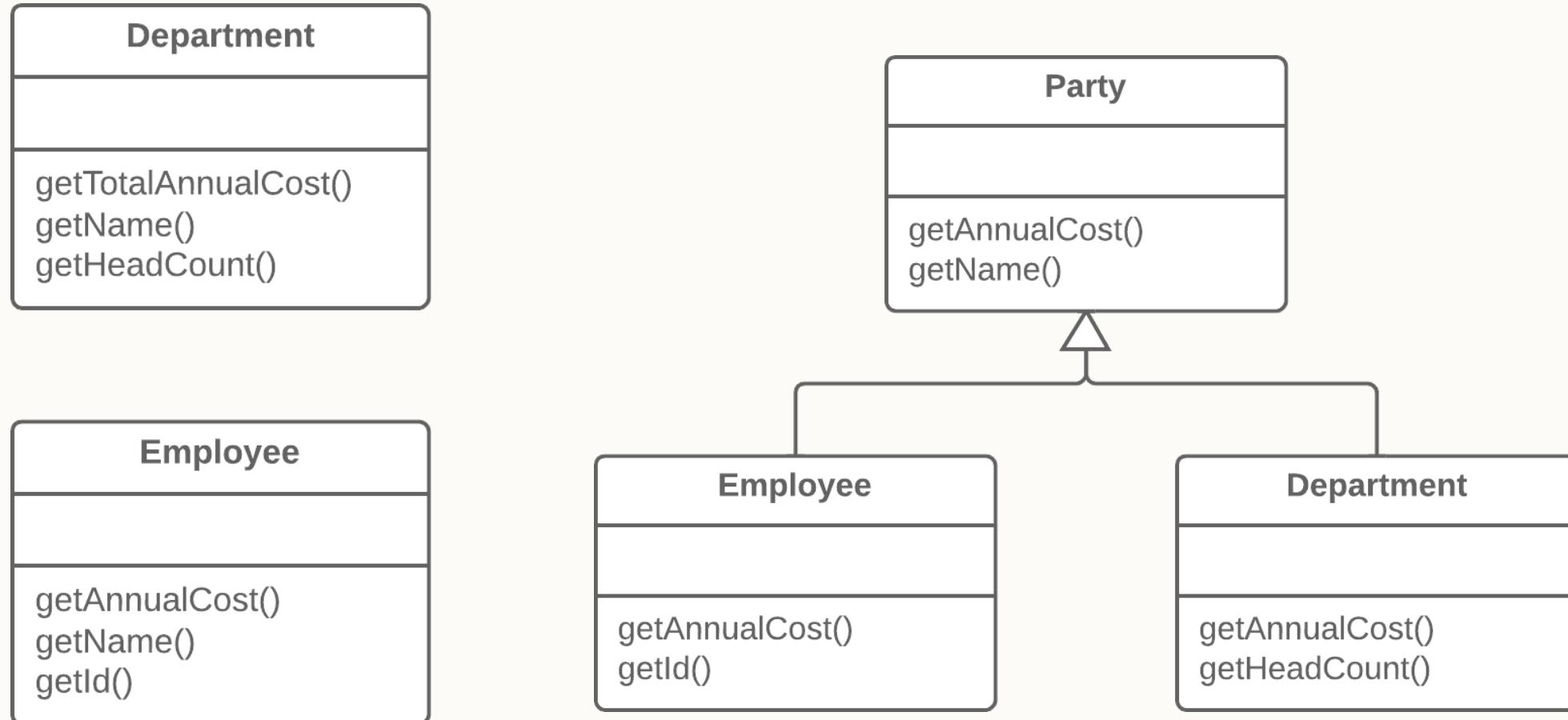
```
public abstract class Person
{
    protected String name;
}

public class Employee extends Person
{
    private String jobTitle;
}

public class Student extends Person
{
    private Course course;
}
```



Calculating annual cost examples





Applying a design pattern in Refactoring

The Form Template Method



Form Template Method adds consistencies to methods so that ‘pull up method’ can be performed

- When you find two methods in subclasses that perform the same steps, but do different things in each step, create methods for those steps with the same signature and move the original method into the base class
- Subclasses are developed in parallel, sometimes by different people, which leads to code duplication, errors, and difficulties in code maintenance
- Each change must be made in all subclasses – *why is this problematic?*
- Code duplication doesn’t always refer to cases of simple copy/paste
 - Often duplication occurs at a higher level, such as when you have a method for sorting numbers and a method for sorting object collections that are differentiated only by the comparison of elements
- Creating a template method eliminates this duplication by merging the shared algorithm steps in a superclass and leaving just the differences in the subclasses



Fowler's Famous Video Rental Example

- In this explanation we make use of the classic example described in the first chapter of Fowler's refactoring [available in Java, Ruby and Javascript]
- Describes a management system for a video rental store –*yes I know that such place don't exist any more... indeed Fowler has changed his recent examples accordingly to involve a theatre company management system.*
- It contains a Movie class, a Customer class and a Rental class, and prints out statements
- Performs a series of refactoring tasks on this original codebase
- Still confused? Here is Martin Fowler himself talking through his example in full using Javascript: <https://martinfowler.com/articles/refactoring-video-store-js/>
- Even more confused? Here's Fowler's lecture:
<https://www.youtube.com/watch?v=6wDoopbtEqk&t=1263s>



Customer Rental System Before Refactoring

Customer

+ string GetStatement();
+ string GetHTMLStatement();

```
public string GetStatement()
{
    var result = new StringBuilder();
    result.AppendLine("Rental record: " + Name);
    foreach (var rental in Rentals)
        result.AppendLine("\t" + rental.Movie.Title);
    result.AppendLine("Owed:" + TotalCharge.ToString());
    result.AppendLine("Customer earned:" +
TotalFrequentRenterPoints.ToString() + "renter points");
    return result.ToString();
}
```



GetHTMLStatement() Is Surprisingly Similar...

```
public string GetHTMLStatement()
{
    var result = new StringBuilder();
    result.AppendLine("<h1>Rental record:<em> " + Name + "</em></h1>");
    foreach (var rental in Rentals)
        result.AppendLine(rental.Movie.Title + "<br />");
    result.AppendLine("<p> Owed: <em>" + TotalCharge.ToString() + "</em></p>");
    result.AppendLine("<p> Customer earned: <em> " +
                    TotalFrequentRenterPoints.ToString() + "</em>renter
points</p>");
    return result.ToString();
}
```



Mechanics of the Form Template Method

1. Decompose the methods so that the extracted methods are identical
2. Apply Pull Up of the identical methods into the the superclass (if using inheritance) or the class that extends the module
3. Test after each pull up/ extraction
4. Use the same set of method calls but the subclasses/modules handle the method calls differently
5. Test again
6. Apply Pull Up to the original method
7. Test again then remove the extraneous methods, test after each removal



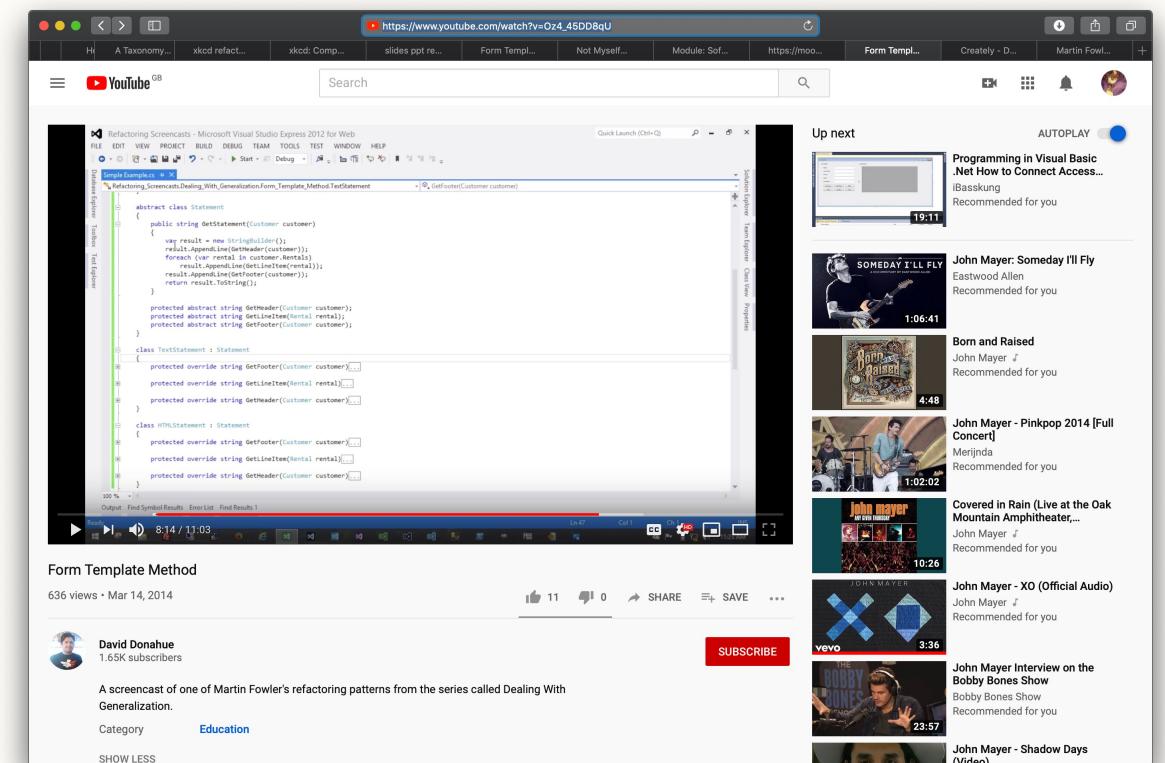
See how it is done, by David Donahue

```
abstract class Statement
{
    public string GetStatement(Customer customer)
    {
        var result = new StringBuilder();
        result.AppendLine(GetHeader(customer));
        foreach (var rental in customer.Rentals)
            result.AppendLine(GetLineItem(rental));
        result.AppendLine(GetFooter(customer));
        return result.ToString();
    }

    protected abstract string GetHeader(Customer customer);
    protected abstract string GetLineItem(Rental rental);
    protected abstract string GetFooter(Customer customer);
}

class TextStatement : Statement
{
    protected override string GetFooter(Customer customer)...
    protected override string GetLineItem(Rental rental)...
    protected override string GetHeader(Customer customer)...
}

class HTMLStatement : Statement
{
    protected override string GetFooter(Customer customer)...
    protected override string GetLineItem(Rental rental)...
    protected override string GetHeader(Customer customer)...
}
```





A note on ‘Big Refactoring’

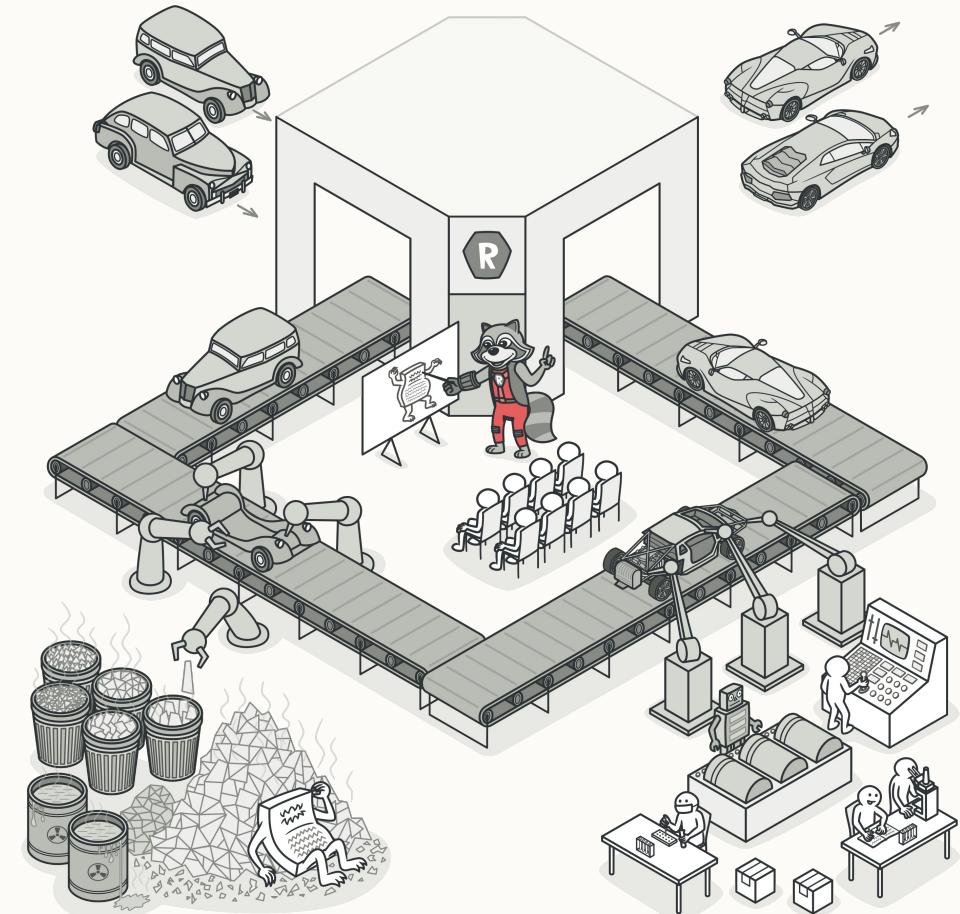
- Little refactoring is about making many almost imperceptible changes to code
- Big refactoring is a team wide effort involving major redesigns
- Four Big Refactorings:
 1. Tease apart inheritance to promote encapsulation, separating out functionality
 2. Convert procedural design features to object oriented (beyond variables) by applying design patterns or SOLID principles
 3. Separate domain from presentation by for example using the MVC
 4. Extract a hierarchy by creating a hierarchy of classes with each subclass representing a specific case
- Massive risk of breaking an entire codebase if careful testing is not applied during these challenging processes – but the payoff in reducing future maintenance costs is huge: *remember, software engineering is about making money from code*



</>

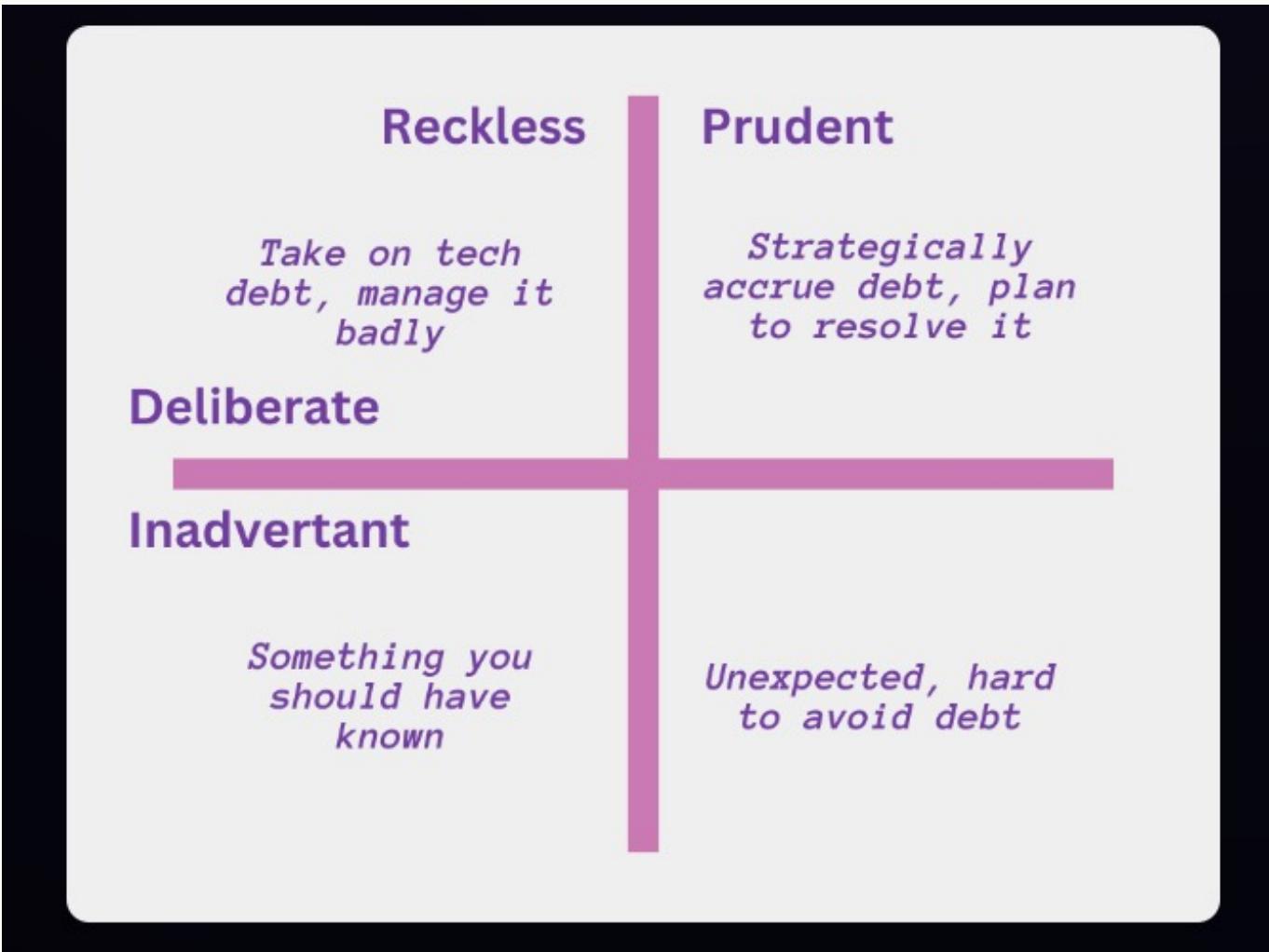
And there are many more to explore

- If you can afford it, buy a copy of Fowler's book – 1st Edition Java, 2nd Ed Javascript
- Check out Refactoring Guru where all of the techniques in the book are explained one by one:
<https://refactoring.guru/refactoring>
- Make sure you watch the video on the Template Method





Technical Debt



Source : <https://www.stepsize.com/blog/types-of-tech-debt-with-examples-and-fixes>

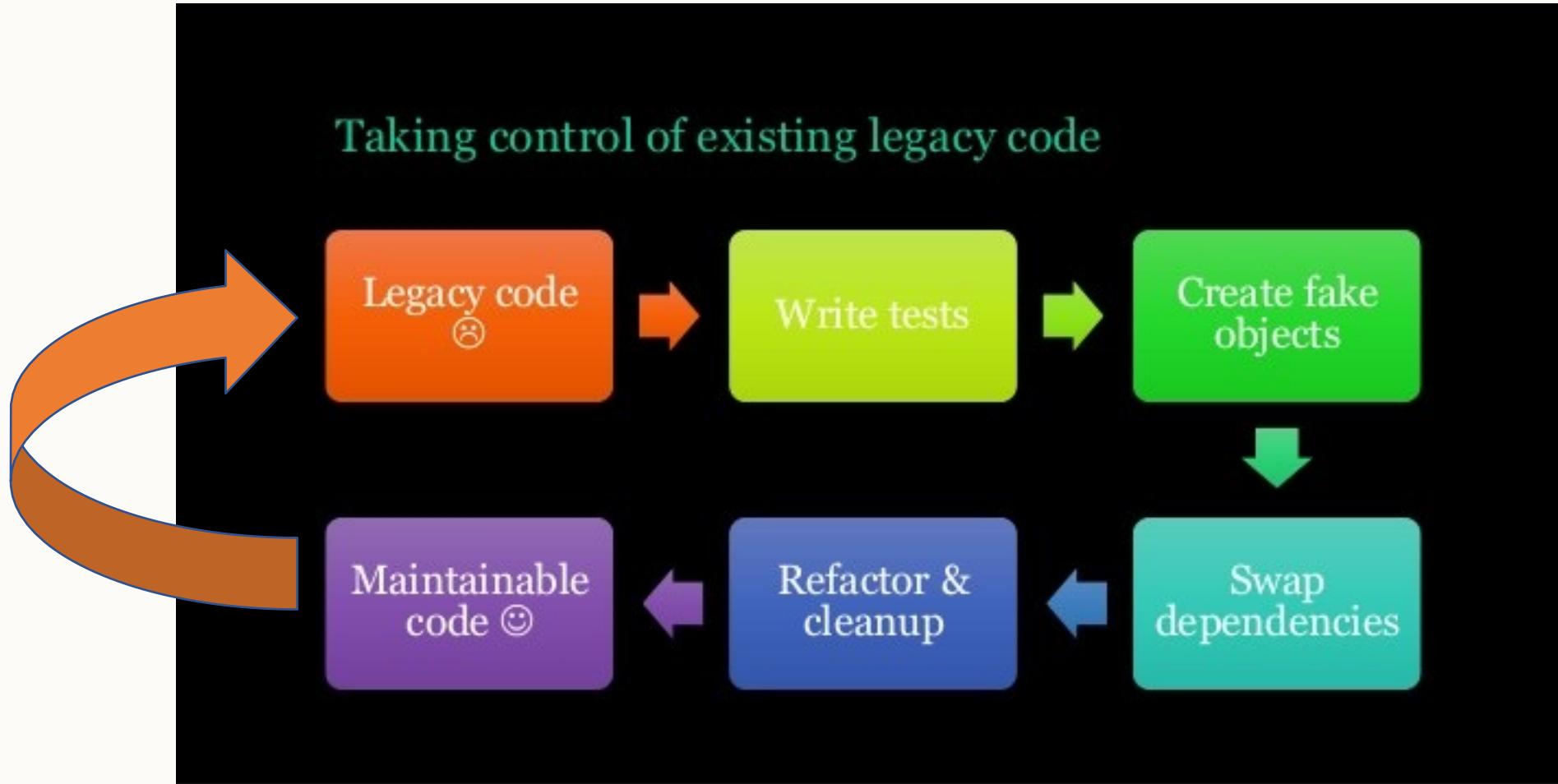


We are doing this to avoid technical debt

- No one sets out to write ‘bad’ code – though the agile principles of minimal design up front and release early and often teach us to focus on delivery rather than clean code
- Technical debt is the same as consumer debt, not following the practice of testing and refactoring over time means that the blob and spaghetti code build up
- You can speed up development without explicit testing but this has consequences
 - Business pressure to release new features all the time
 - Monolithic code rather than decomposed modules
 - Lack of testing and documentation
 - Lack of interaction between team members
 - Lack of compliance monitoring



As a recap, this is an iterative process





</>

Self Study Quick Questions

Legacy code is ...

Code smells are ...

Refactoring is ...

Testing works with refactoring because...

The extract method works by...

Replace magic numbers with...

The Form Template Method uses...



Enjoy Refactoring Week

- Refactoring and legacy code
- Principles and motivation for refactoring
- Martin Fowler and Kent Beck
- Small scale refactoring
- Object orientation and SOLID
- Applying patterns
- Many more refactoring techniques....



Today: Lecture on Refactoring

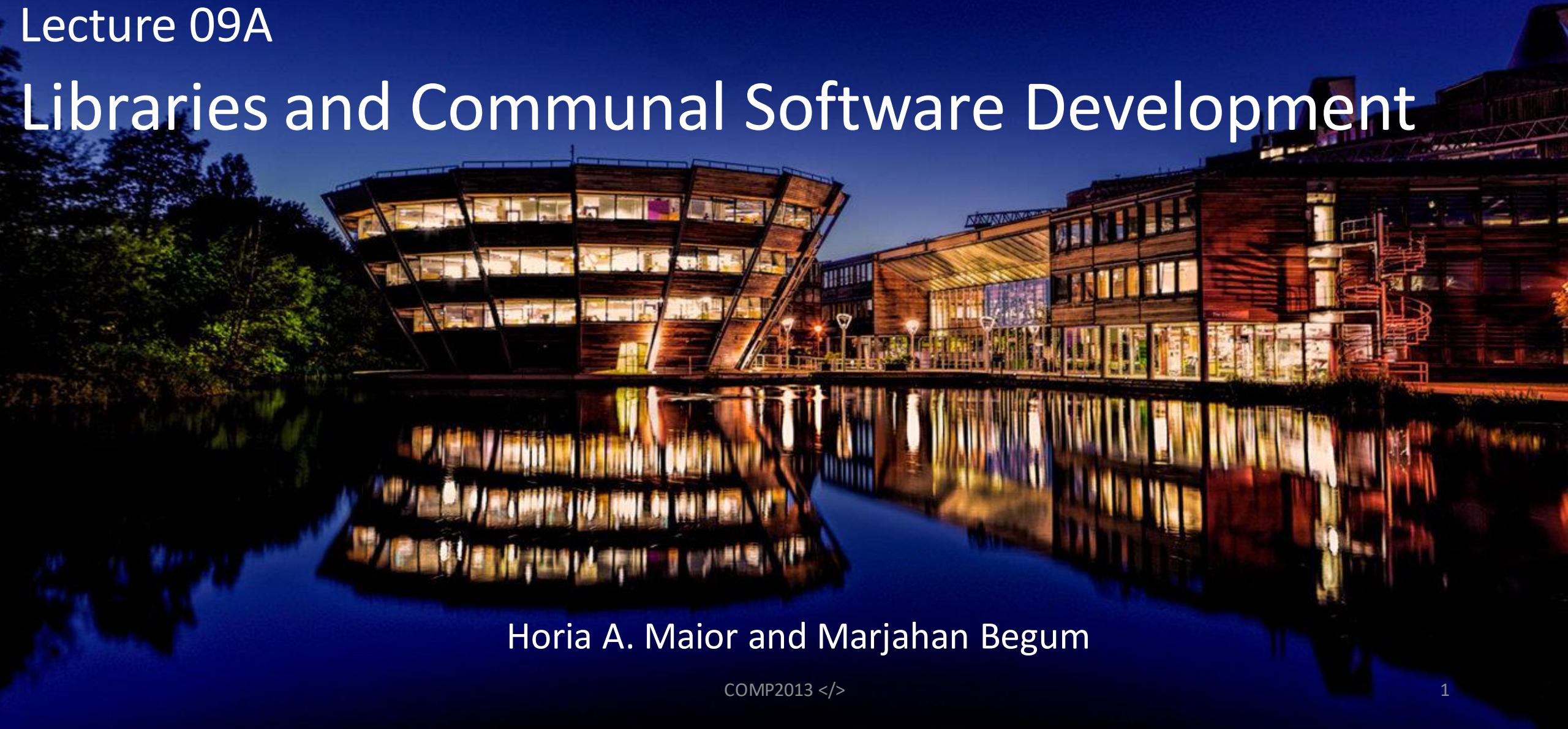


Friday 24th November : Mileston1
Coursework deadline, 5pm



Lecture 09A

Libraries and Communal Software Development



Horia A. Maior and Marjahan Begum

</>



Lecture 09A

Libraries and Communal Software Development

Horia A. Maior and Marjahan Begum

- Lecture
 - Coursework tips
 - Using and creating libraries
 - Communal software development (what it is; licensing; how to do it; how to get involved)
- Lab
 - Time for you to work on your coursework
- Lecture
 - Time for you to work on your coursework

coursework tips

Screen Capture Demos on Moodle

</>

There are many useful screen capture demonstration videos in the lectures and on Moodle:

- Setting Up IntelliJ
- Setting Up Git and GitLab
- Maven/Gradle Project
- JavaFX Project
- Setting Up Junit testing within a JavaFX project using Maven (later today)
- Early set up of the CW

Path Issues

- An error message like this indicates that the compiler cannot find an image
 - Usually the path is set wrongly

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "java.net.URL.toString()"
  at java.desktop/sun.awt.SunToolkit.getImageFromHash(SunToolkit.java:698)
  at java.desktop/sun.awt.SunToolkit.getImage(SunToolkit.java:734)
  at snakeeProj.main@1.0-SNAPSHOT/snakee.MyFrame.<init>(MyFrame.java:32)
  at snakeeProj.main@1.0-SNAPSHOT/snakee.Play.<init>(Play.java:9)
  at snakeeProj.main@1.0-SNAPSHOT/snakee.Play.main(Play.java:63)
```

```
30     public MyFrame()
31     {
32         jFrame.setIconImage(Toolkit.getDefaultToolkit().getImage( filename: "snake-logo.png"));
33     }
```

- Likely to do with the same issue the paths to resources folder.

</>

```
java.lang.IllegalArgumentException Create breakpoint : input == null!
    at java.desktop/javax.imageio.ImageIO.read(ImageIO.java:1400)
    at CW1/cw1_files.GameUtil.getImage(GameUtil.java:19)
    at CW1/cw1_files.ImageUtil.<clinit>(ImageUtil.java:14)
    at CW1/cw1_files.MyFrame$MySnake.<clinit>(MyFrame.java:109)
    at CW1/cw1_files.Play.<init>(Play.java:22)
    at CW1/cw1_files.Play.main(Play.java:69)
VILLA : FINN EKKI TILTEKNA MYNDIN !
```

```
java.lang.IllegalArgumentException Create breakpoint : input == null!
    at java.desktop/javax.imageio.ImageIO.read(ImageIO.java:1400)
    at CW1/cw1_files.GameUtil.getImage(GameUtil.java:19)
    at CW1/cw1_files.ImageUtil.<clinit>(ImageUtil.java:15)
    at CW1/cw1_files.MyFrame$MySnake.<clinit>(MyFrame.java:109)
    at CW1/cw1_files.Play.<init>(Play.java:22)
    at CW1/cw1_files.Play.main(Play.java:69)
VILLA : FINN EKKI TILTEKNA MYNDIN !
```

```
java.lang.IllegalArgumentException Create breakpoint : input == null!
    at java.desktop/javax.imageio.ImageIO.read(ImageIO.java:1400)
    at CW1/cw1_files.GameUtil.getImage(GameUtil.java:19)
    at CW1/cw1_files.ImageUtil.<clinit>(ImageUtil.java:17)
    at CW1/cw1_files.MyFrame$MySnake.<clinit>(MyFrame.java:109)
    at CW1/cw1_files.Play.<init>(Play.java:22)
    at CW1/cw1_files.Play.main(Play.java:69)
VILLA : FINN EKKI TILTEKNA MYNDIN !
```

```
java.lang.IllegalArgumentException Create breakpoint : input == null!
    at java.desktop/javax.imageio.ImageIO.read(ImageIO.java:1400)
```

Maven. How to use a POM file?

- Project Object Model (POM)
- Contains configuration information about your project
- Key items
 - Project Identification
 - Dependencies
 - Plug-Ins
 - Other Settings

Error: Reading packages from multiple sources

</>



The screenshot displays the IntelliJ IDEA Project Structure dialog. The left sidebar shows 'Project Settings' with 'Libraries' selected. In the main pane, a library named 'googlecode.soundlibs.jlayer' is shown, configured with Maven coordinates 'com.googlecode.soundlibs:jlayer:1.0.1.4'. Below it, the 'Classes' section lists two JAR files from the local Maven repository:

- C:\Users\pszpsadmin\.m2\repository\com\googlecode\soundlibs\jlayer\1.0.1.4\jlayer-1.0.1.4.jar
- C:\Users\pszpsadmin\.m2\repository\junit\junit\3.8.2\junit-3.8.2.jar

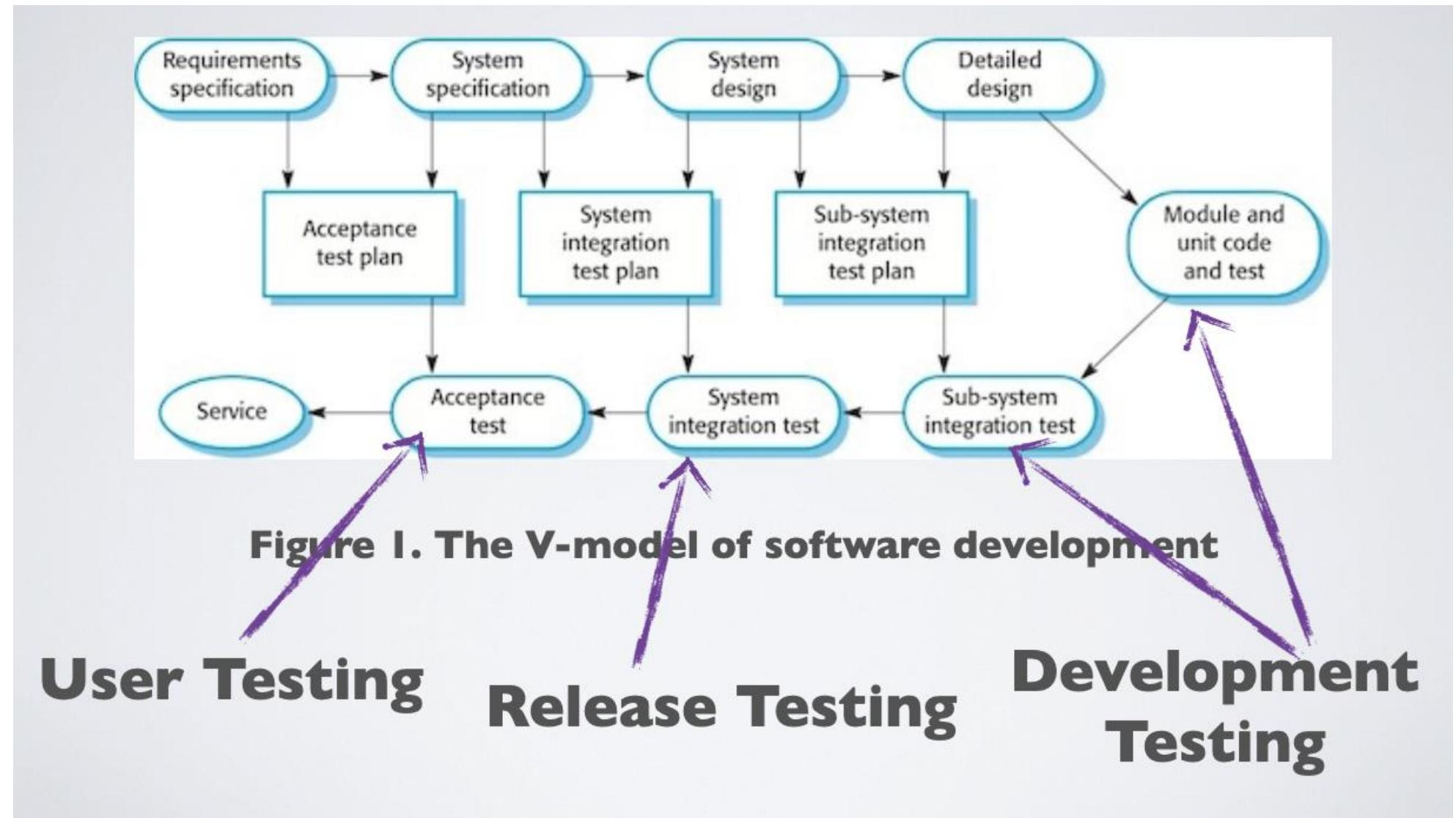
- Consider the use of "streams" and "lambdas" for improving maintainability
 - <https://stackify.com/streams-guide-java-8/>
- Solving problems is detective work
 - Check out "Stack Exchange"
 - Use prototyping and throw away models
 - Use the debugger to get information about the state of variables
 - https://youtube.com/playlist?list=PLPZy-hmwOdEUWF85MuwrKV8YVWLmZW4ZA&si=gj8AmbZVC9q_xQMY
 - Look at the examples from the lectures and on Moodle
- When submitting your coursework
 - Don't forget to merge back to the main branch
 - Don't forget to tidy up your project by using "mvn clean" or "gradle clean" before zipping it

Random Tips

</>

- Feel free to use your own graphics/sound
- Testing plan/strategy
 - White Box
 - Black Box
 - Keep Track of Testing

Black Box Testing	White Box Testing
The Black Box Test is a test that only considers the external behavior of the system; the internal workings of the software is not taken into account.	The White Box Test is a method used to test a software taking into consideration its internal functioning.
It is carried out by testers.	It is carried out by software developers.
This method is used in System Testing or Acceptance Testing .	This method is used in Unit Testing or Integration Testing .
It is the least time consuming.	It is most time consuming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is also known as data-driven testing, functional testing , and closed box testing.	It is also known as clear box testing, code-based testing, structural testing, and transparent testing.
Black Box Test is not considered for algorithm testing.	White Box Test is well suitable for algorithm testing.



User Testing

Release Testing

Development Testing

System Tests Document

</>

Test ID	Reason	Input	Expected Output	Pass/Fail
1		-1	error	PASS (date)
2	A description of what we are testing	4.99	converted to 5	PASS (date)
3		10,000,000,000,000	error	1) FAIL (date) 2) PASS (date)
4	Another thing to test	-1	error	PASS (date)

Useful Resources

</>

- Maven tutorial Crash Course <https://www.youtube.com/watch?v=Xatr8AZLOsE>
- Gradle tutorial Crash Course <https://www.youtube.com/watch?v=gKPMKRnnbXU>
- JUnit 5 tutorial Crash Course <https://www.youtube.com/watch?v=6uSnF6luWIw>

philosophies of software development

Different Philosophies of Software Development

</>

- During your life as a software maintainer you will work on a number of different forms of projects
- Some of these may have open source code, or use open source libraries
- You may have to rework private code to be open source, or vice versa
- In this lecture, we will look at an overview of how to use third party code in the form of libraries and open source projects

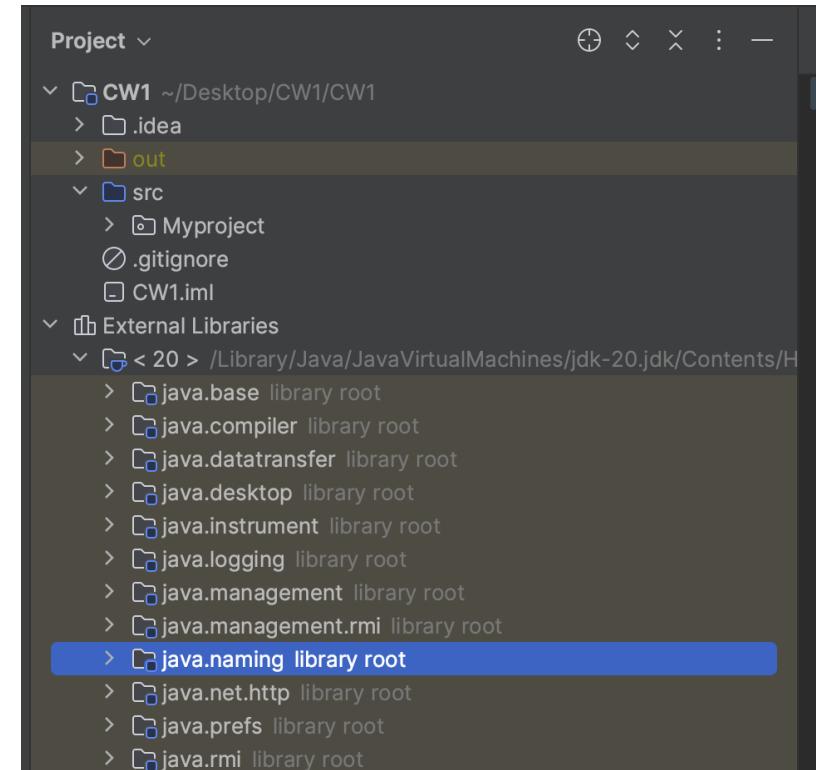
</>

libraries

What is a Library?

</>

- What is a library?
 - Some 3rd party software packaged up (in binaries) and ready-to-use in your own code
 - It is a shared resource
- Usually online documentation
 - Supporting guides as well as Javadocs to show the API
- You've already had experience of this
 - e.g. using the JDK library in your IDE



What's in a Library?

</>

- Libraries in Java consists of:
 - A Jar file
 - Created via the jar tool, or via an IDE
 - Basically a zip file
 - A way of packaging class files and resource files
 - Contains a special folder – META-INF
 - Note that a runnable Jar file is not a library!
 - An API
 - Publicly accessible methods
 - Interface stability is important; use "@Deprecated" before removing methods
 - In Javadoc use "@deprecated"
 - Usually include a licence
 - How you can distribute/change it – more later

Making Use of Libraries in Java

</>

- You need a library file - usually a .jar archive
- Reference it in your project
 - e.g. adding the jar to your Java Build Path
- Import relevant parts of the library into your code
- Make use of the methods
 - May need to create an object, or static use access
- Think about how to distribute the library with your code

Making Use of Libraries in Java

</>

- You can package them up with your deployed application, or perhaps include them in your project source distribution
 - Licence permitting! (more later...)
- You can also use build files to help with collaborative development using libraries
 - Script will copy a file from an external resource
 - In this way you do not need a license to re-distribute them as you are only downloading them

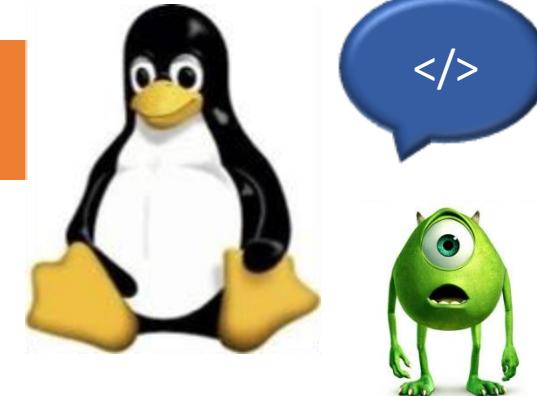
Build Systems and Libraries

</>

- Build systems can pull required libraries from remote sources
- Maven/Gradle will place these dependences in a folder locally:
{your-username}\.m2
{your-username}\.gradle
- You can request particular versions of libraries, so you may see different versions of libraries in this folder

open source software development and maintenance

Open Source Software



- What is Open Source Software (OSS)?
 - OSS is (generally) free software that uses any licence approved by the Open Source Initiative (OSI) from their list of approved open source licences (link below)
- What is Free OSS?
 - "Software that gives users rights to run, copy, distribute, change and improve it as they see it, without them asking permission from or make payments to any external group or person".
- Open source initiative: <https://opensource.org/>

Mitre FOSS report 2002



- Richard Stallman: President and founder of the Free Software Foundation



"Free software should be a human right!"

Different perspectives

Free software activists



Open source boosters



Free software, free society: Richard Stallman at TEDxGeneva 2014
https://www.youtube.com/watch?v=Ag1AKII_2GM

Open Source Software

</>

- Why Go Open Source?
 - Higher Quality
 - Customisable
 - Improvable
 - Collaborative bug finding/fixing
 - Redistributable
 - Transparency
 - Free

Open Source Software

</>

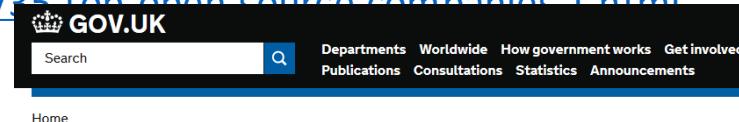


- Starting to be recognised as a "Good Thing" by industry. Why?
 - All the previous reasons plus:
 1. Good advertising
 2. Attract talented developers
 - a) More development possible
 - b) Version-tracked contributions are good demonstration of potential employees' programming skill
 - c) Develop outside of your core skill set

Open Source Software

</>

- Examples:
 - Linux, OpenJDK; Apache; LaTex; Moodle; Firefox; Android; Mozilla; MySQL; OpenOffice; Blender; VLC; IntelliJ CE; Eclipse
- Top "Open Source" Companies
 - Adobe; Amazon Web Services; Docker; Facebook; GitHub; Google; Gradle; Huawei; IBM; Intel; LinkedIn; Microsoft; MongoDB; Netflix; Oracle; Red Hat; Samsung Electronics; Twitter;
...
 - More information about how these companies contribute is available here:
 - https://www.datamation.com/open-source/35_top_open_source_companies_1.html
- The UK government supports OSS,
 - They should do!



Guidance **Be open and use open source**

Publish your code openly and use open source technology to improve transparency, flexibility and accountability.

Open Source Definition

</>

- Open Source Criteria:
 - <https://opensource.org/osd/>



open source
initiative®

About Programs Licenses **Open Source Definition** News Join

Home / The Open Source Definition

The Open Source Definition

Page created on July 7, 2006 | Last modified on February 22, 2023

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source

- Range of Code Adoption

- Code adoption can happen at the level of a few lines of code, a method, a class, a library, a component, a tool, or a complete system.

The Ethics of Open Source

</>

- Different points of view:
 - The ethics of free software (Dr Dobb's Journal 2000)
 - <http://www.drdobbs.com/the-ethics-of-free-software/184414581>
 - The ethics of open source software (Erfanian's Blog 2013)
 - <http://www.ericerfanian.com/the-ethics-of-open-source-software/>
 - Why open source software isn't as ethical as you think it is (Ethical Tech 2017)
 - <https://words.werd.io/why-open-source-software-isnt-as-ethical-as-you-think-it-is-2e34d85c3b16>

UoN - Responsible Research and Innovation

</>

<p>Act</p> <p>In the AREA-4Ps framework the fourth key activity is to <i>Act</i>; that is to use the insights gained from anticipation, reflection and engagement in order to <i>make a difference</i> in the work being done. This closes the loop of responsible innovation: ultimately, responsibility can only be discharged through action.</p> <p>Within the deck, every card includes a number of example actions. These lists are not exhaustive, and there are many other resources and practices available to support responsible innovation. In this deck the <i>Act</i> cards look beyond the current project.</p> <p>2023-06-02</p>	<p>Shaping the Future</p> <p>How can we shape a better future for everyone? How can we reduce inequalities? What can we contribute to regulation & legislation?</p> <p>Example actions: 2023-06-02</p> <ul style="list-style-type: none">• Talk to policy makers.• Respond to requests for evidence from government, regulatory and public bodies.• Run a publicity or impact campaign.• Contribute to professional bodies and standards. <p>Act Purpose</p>	<p>Openness</p> <p>How can others build on the work done? Is support available for this? Is all relevant information disclosed? Are publications and reports widely available? Is data appropriately archived?</p> <p>Example actions: 2023-06-07</p> <ul style="list-style-type: none">• Be transparent about the work and any products.• Publish and publicise the outcomes.• Make data FAIR (Findable, Accessible, Interoperable, Reusable).• Adopt open licenses.• Support adoption by others. <p>Act Product</p>	<p>Training and Equipping</p> <p>What training and support do team members need? How do we help participants and partners to grow and develop? How do we support formal and informal education?</p> <p>Example actions: 2023-06-08</p> <ul style="list-style-type: none">• Provide tailored support and training for team members and other stakeholders.• Develop an education or outreach plan.• Contribute to local public engagement events.• Continue to engage with stakeholders afterwards. <p>Act People</p>	<p>Continuous Improvement</p> <p>What actions can we take throughout this project to improve ourselves, the work and our organisation? What can we learn from this and previous projects? How can we support RI more effectively?</p> <p>Example actions: 2023-06-08</p> <ul style="list-style-type: none">• Share resources and ideas with peers.• Hold periodic reviews.• Proactively raise issues at an appropriate level, e.g. project, department, organisation.• Recruit strategically.• Champion responsible innovation. <p>Act Process</p>
---	--	--	--	---

<https://tas.ac.uk/responsible-research-innovation/rri-prompts-and-practice-cards/>

licences



- I just want to give my software away! Why do I care about licences?
- Scenario 1
 - Bob installs my disk optimisation software. Great! But Bobs hard drive then catches fire and he loses an entire novel he is writing. Bad. Who's fault is it?
- Scenario 2
 - I write an awesome music sharing app. Great!
 - EvilCorp also like it, and they realise they can take it, close the source, and fill it with ads and sell it. Is this what I want?



Software Licences

</>

- Software licences are there ...
 - to protect you as well as your code
 - to protect any future developers of the code
- We need rules in order to secure certain freedoms
 - What can be done with the code
 - Who can change it
 - Who can distribute it
 - Is there any warranty or disclaimer
 - ...

- Some common OSS licences (for more see <https://opensource.org/licenses/> and https://en.wikipedia.org/wiki/Software_license)
 - Permissive license:
 - Subsequent users can produce 'closed source' versions and sell the software
 - CopyLeft license:
 - Any subsequent versions are left with the same rights e.g. source code must be supplied, and can be modified
- Examples
 - Apache Licence 2.0 > Permissive licence
 - BSD 3-Clause "New" or "Revised" licence > Permissive licence
 - GNU General Public Licence (GPL) > CopyLeft licence
 - MIT licence > Permissive licence
 - Mozilla Public Licence 2.0 > Permissive licence

what next?

Using and Developing Your Skills

</>

- What have you learnt during the COMP2013 module?
 - Much more experience with Java
 - Object-Oriented Design
 - GUI programming
 - Use of tools (e.g. Git/GitLab; IntelliJ; Maven/Gradle; ...)
 - Modifying and adding to a sizeable existing project
- And with your group project you are learning to work as a team
- What now? You might want to:
 1. Initiate your own open source project
 2. Contribute to an existing project

Initiate Your Own Project

</>

- How?

- Identify a need
- Discuss the idea
- Does it already exist?
- Talk to your lecturers / peers
- Code Development starts
- Create website and "announce"
- Community evolves
- Functionalities added

- Funding?

- Donation, voluntary, crowd-funded



Contribute to an Existing Project

</>

The screenshot shows the GitHub repository page for `freeciv/web`. The repository has 5,551 commits. The 'About' section describes Freeciv-web as an Open Source strategy game implemented in HTML5 and WebGL, which can be played online against other players, or in single player mode against AI opponents. It includes links to Readme, View license, Activity, 1.9k stars, 84 watching, 324 forks, and Report repository. The 'Releases' section shows one release, 'Release of Freeciv-web to pr...', dated Jul 23, 2015. The 'Packages' section indicates no packages published. The 'Contributors' section lists 34 contributors with small profile icons. The 'Languages' section shows the following distribution: JavaScript (73.4%), Java (9.1%), Python (6.8%), CSS (6.0%), Shell (2.5%), GLSL (1.4%), and Other (0.8%).

<https://github.com/freeciv/freeciv-web>

<https://github.com/freeciv/freeciv-web/issues>

Contribute to an Existing Project

</>

- How to get involved
 - Look at the README file
 - Should explain the purpose of the projects, direction of development, etc.
 - Look out for sections on How To Contribute
 - Fork the project
 - Create a branch?
 - Check for any rules on how contributors should work
 - Respect the rules of the project

Contribute to an Existing Project

</>

- How to get involved (cont.)
 - Join a development chat or forum to see how development is organised
 - Start to talk to the developers if you think you can help
 - Learn how to use merge/pull requests
 - Write a test to show you have fixed a bug
 - Have the right attitude!
 - Polite; patient

Contribute to an Existing Project

</>

- Looking for open issues
 - Look under Issues for a project
 - Projects can assign labels
 - Some target new developers

<p> ⓘ Make sure that jest tests fail if an error is thrown within a jsdom event handler good first bug #8260 opened 20 days ago by spicyj</p>
<p> ⓘ [New Docs] Wanted Guides Component: Documentation & Website good first bug #8060 opened on 23 Oct by gaaron 0 of 13</p>
<p> ⓘ how remove debug info from react.min.js file when publish project good first bug #7990 opened on 17 Oct by uxitten</p>
<p> ⓘ Show component stack for invalid type warning during element creation good first bug #7856 opened on 4 Oct by spicyj</p>
<p> ⓘ Make createElement(undefined) warning more descriptive good first bug Type: Enhancement #7307 opened on 19 Jul by gaaron</p>
<p> ⓘ Should React warn when controlled <select> components have duplicate values? good first bug Type: Enhancement #6959 opened on 3 Jun by jbinto</p>

Code Issues Pull req

- ⓘ Add Git & GitHub Challenges curriculum request
#11515 opened 26 days ago by atjonathan
- ⓘ Question about challenge: Label Bootstrap Buttons. Discussing enhancement
#11477 opened on 31 Oct by zhuxiang19910319
- ⓘ Read-Search-Ask links broken all over the place blocked
#11465 opened on 30 Oct by Kwpolska
- ⓘ Progress gone after clicking activation link twice accounts blocked
#11462 opened on 30 Oct by RichStone

Contribute to an Existing Project

</>

- Communication is informal
 - Community communications
 - Threaded discussion forums
 - Email (list servers)
 - Newsgroups
 - Messaging/chat
 - Community digests
 - Social networks

Fix region contain method #364

 Open mickare wants to merge 2 commits into sk89q:master from mickare:patch-1

Conversation 14 Commits 2 Files changed 1

mickare commented on 13 Sep

The region contain method was broken.
A lot of subsequent issues are caused by this bug.
E.g. in a selection the entities in the last blocks at the positive-axis border are not selected.

Max block gives the impression of an exclusive point; however it is inclusive!
A position that is anywhere between of a 1x1x1 region, would return false in the old implementation.

By simply adding 1 should solve the problem.

Greetings,
mike

Fix region contain method ... ✓ 5de4fcf

TomyLobo commented on 13 Sep Collaborator

Nice spot.
The difference between `((double) a) <= ((int) b)` and `((double) a) < ((int) b) + 1` is indeed significant.
However, wouldn't it be better to just use `getBlockX / Y / z` to obtain `x / y / z` and turn them into `int s?`

Challenges of Collaborative OSS Development

</>

- Product structure and comprehension
 - Who understands the 'whole system'?
- Effective ways of incorporating requirements of non-developer users?
- With larger scales, will coordination needs force adoption of "commercial" development techniques?
- How to collaborate on "big" features?
- How to respond to unanticipated events? (Funding etc.)

and finally ...

Reading for First Timers: <http://www.firsttimersonly.com/>

GitHub for Beginner's Guides and Help: <https://github.com/btford/participating-in-open-source/>

Acknowledgements

</>

Thanks to Peer-Olaf Siebers and Robert Laramee for the lecture materials

JUnit refresher

```
22      </dependency>
23      <dependency>
24          <groupId>org.openjfx</groupId>
25          <artifactId>javafx-fxml</artifactId>
26          <version>21.0.1</version>
27      </dependency>
28      <dependency>
29          <groupId>org.junit.jupiter</groupId>
30          <artifactId>junit-jupiter-engine</artifactId>
31          <version>5.8.2</version>
32      </dependency>
33
34
35      </dependencies>
36
37      <build>
38          <plugins>
39              <plugin>
40                  <groupId>org.apache.maven.plugins</groupId>
41                  <artifactId>maven-compiler-plugin</artifactId>
42                  <version>3.11.8</version>
43                  <configuration>
44                      <source>20</source>
45                      <target>20</target>
46              </plugin>
47          </plugins>
48      </build>
49  
```

project dependencies dependency version

new/jek-20.jdk/Contents/Home/bin/java ...

e 8

[JUnit Test - HD 1080p.mov](#)



Lecture 10 – Important Miscellaneous

COMP2013 (AUT1 23-24)

Dr Marjahan Begum and Dr Horia A. Maior



Register your attendance

COMP2013: Developing Maintainable Software
Week 11 – 4:00pm Monday – 04 December 2023



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



Content

- Module Feedback
- Milestone 1 - High Level Feedback
- Milestone 2 – Tips
- Exam Revision: Important Topics
- Exam Formalities
- Mock Exam
- Some Final Words



Module Feedback



Can all of you please participate ...

- <https://bluecastle-uk-surveys.nottingham.ac.uk>
- Log in using username and password
- All views are welcome
 - Positive
 - Negative
 - Neural
- Survey starts : 04/12/2023 09:00
- Survey ends : 08/12/2023 17:00





Coursework Feedback

High level Feedback



Milestone1 High Level Feedback

- Provisional marks will be released with feedback
 - Friday 8th December
- Any queries regarding the coursework
 - Please ask us during Friday's lab session



Git

- Most people set up git and number of commits and useful messages
- Some only showed that they have updated the repo with the current code base
- Very good use of branches
 - Main branch
 - Different development branches
- Too much time on Git



UML Class Diagrams

- Initial diagram and automatic class diagram
 - Clear explanations of the individual classes and details about the relationships
 - Brief comparison is made with automatic and self created
- Revised class diagram
 - OO principles applied
 - Clear explanations of the individual classes and details about the relationships
 - Rationalized why it's a better class diagram
 - Some discussed about patterns they will employ



Understanding of current software structure and refactoring

- Main classes in the source code are explained
 - In relation to the class diagram
- Running the game
- Initial refactoring were shown
- Some ideas about future refactoring in relation to OO principles and design patterns



Video quality

- Disproportionate time spent on git
- Speaking too fast
- Image poor quality
- Not explaining what you are showing just say this is initial and this is revised
- No audio



EXAM Revision

Important topics



Fowler Defined Different Categories of Refactoring

- *Composing Methods*: refactoring within a method or within an existing class
- *Moving Features Between Objects*: refactoring changing the responsibility of a class
- *Organizing Data*: refactoring to improve data structures and object linking
- *Simplifying Conditional Expressions*: encapsulation of conditions by replacing with polymorphism
- *Making Method Calls Simpler*: refactorings that make interfaces simpler
- *Dealing with Generalization*: moving methods up and down the hierarchy of inheritance by (often) applying the Factory or Template design pattern
- *Big Refactorings*: architectural refactoring to promote loose coupling or to realise a redesign via object orientation (Note: this is extremely difficult for most projects).



Software Maintenance (SWM)

- What is it?
- Why is it important?
- Different types?
- What are the challenges?

Maintainability is not an afterthought, but should be addressed from the very beginning of a development project



Maintainable Software

- Overview of generic maintainability guidelines:
 - Write short units (constructors/methods) of code
 - Write simple units of code
 - Write code once
 - Keep unit interfaces small
 - Separate concerns in modules (classes)
 - Couple architecture components loosely
 - Keep your codebase small
 - Automate development pipeline and tests
 - Write clean code



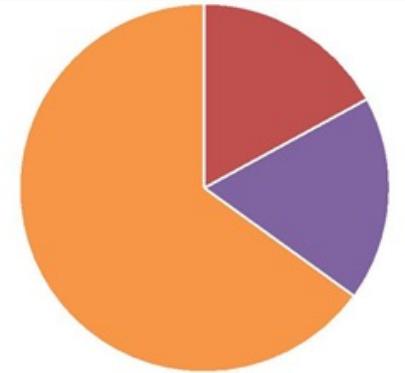
Visser et al (2016)



Three (or four, depending on authors) Main Categories of SWM

- Corrective Maintenance
 - Finding and fixing errors in the system
 - e.g. bugs
- Adaptive Maintenance
 - The system has to be adapted to changes in the environment in which it operates
 - e.g. VAT change, bank offers new mortgage product
- Perfective + Preventive Maintenance
 - Users of the system (and/or other stakeholders) have new or changed requirements
 - Ways are identified to increase quality or prevent future bugs from occurring

Maintenance effort

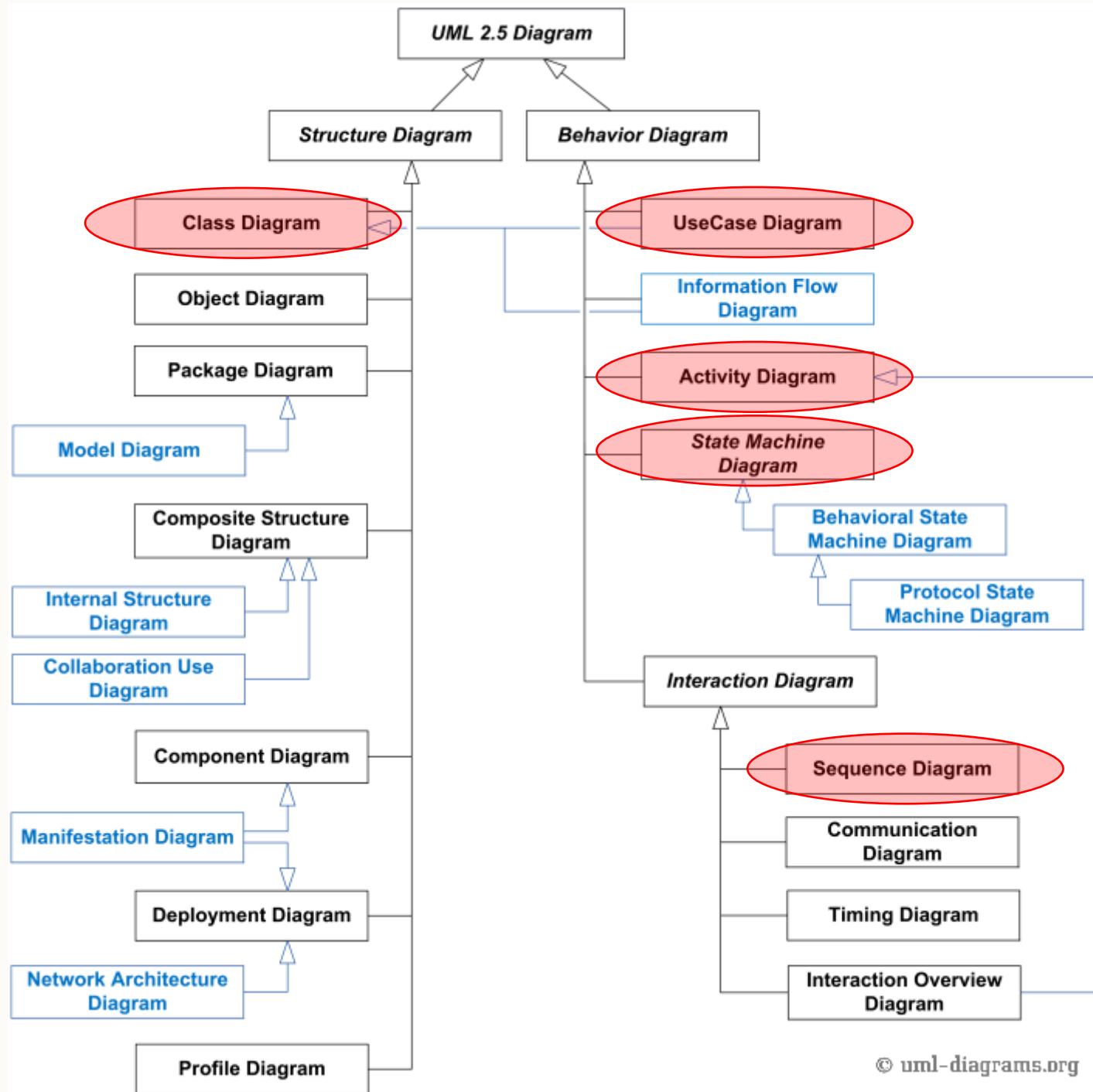


■ Corrective ■ Adaptive ■ Perfective



UML and its Java Implementation

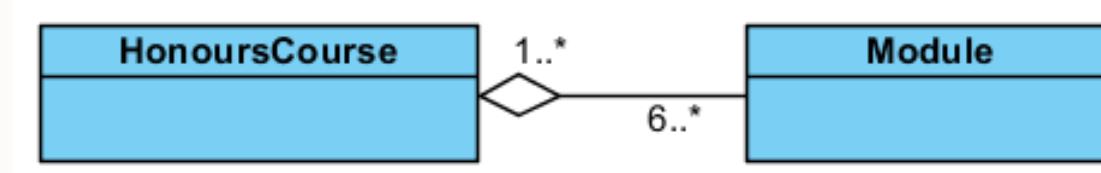
- UML diagrams
 - You only need to consider the ones we discussed in the lecture
 - What are they used for?
- Class diagrams
 - Relationship types and multiplicity
 - Design
 - Implementation





Class Diagram: Design

- Aggregation ("is part of" relationship)

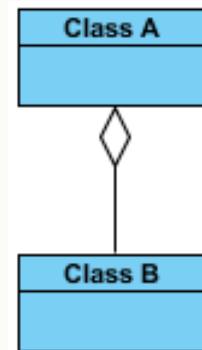


Each Honours Course consists of 6 or more Modules; each Module could be part of one or more Honours Courses



Class Diagram: Implementation

- Class B **is part of** class A (semantically) but class B can be shared and if class A is deleted, class B is not deleted.
 - Class A stores the reference to class B for later use
 - Often setter injection is used



```
2 public class A {  
3  
4     private B b;  
5  
6     public void setB(B b) {  
7         this.b = b;  
8     }  
9 }  
10 }
```



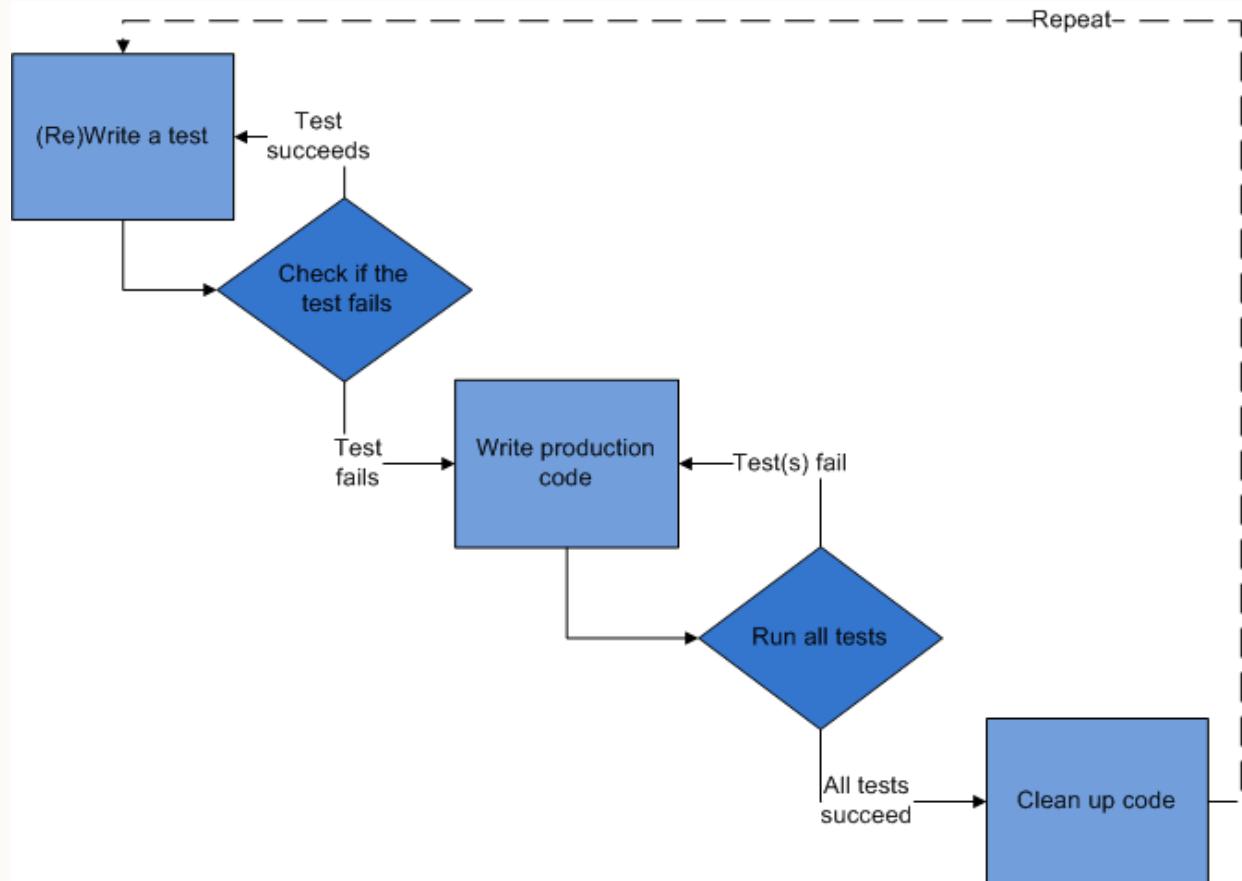
Coding Tools for SWM

- Documenting code
 - Javadocs
 - Make sure you remember the common block tags (e.g. @param)
 - Build files (build scripts)
 - Common tools (e.g.; Maven; Gradle)
 - Why build files are useful
 - Typical tasks accomplished with build files
- Testing
 - Unit and Regression Testing: What's the difference?
 - Unit Testing
 - Test-Driven Development (TDD)



Test-Driven Development

- How does TDD work?
 - Write a test
 - Check if test fails
 - Test fails > Write production code
 - Run all tests
 - All tests succeed > Clean up code
- What does writing the test first allow us to do?
 - Test can be derived from requirements, so it makes sure we meet the specs
 - We write the minimum amount of code to pass (following XP's YAGNI principle)
 - We write maintainable code





Test-Driven Development

JUnit 4 ▾ Project Documentation ▾

Best Practices

When should tests be written?

Tests should be written before the code. Test-first programming is practiced by only writing new code when an automated test is failing.

Good tests tell you how to best design the system for its intended use. They effectively communicate in an executable format how to use the software. They also prevent tendencies to over-build the system based on speculation. When all the tests pass, you know you're done!

Whenever a customer test fails or a bug is reported, first write the necessary unit test(s) to expose the bug(s), *then* fix them. This makes it almost impossible for that particular bug to resurface later.

Test-driven development is a lot more fun than writing tests after the code seems to be working. Give it a try!

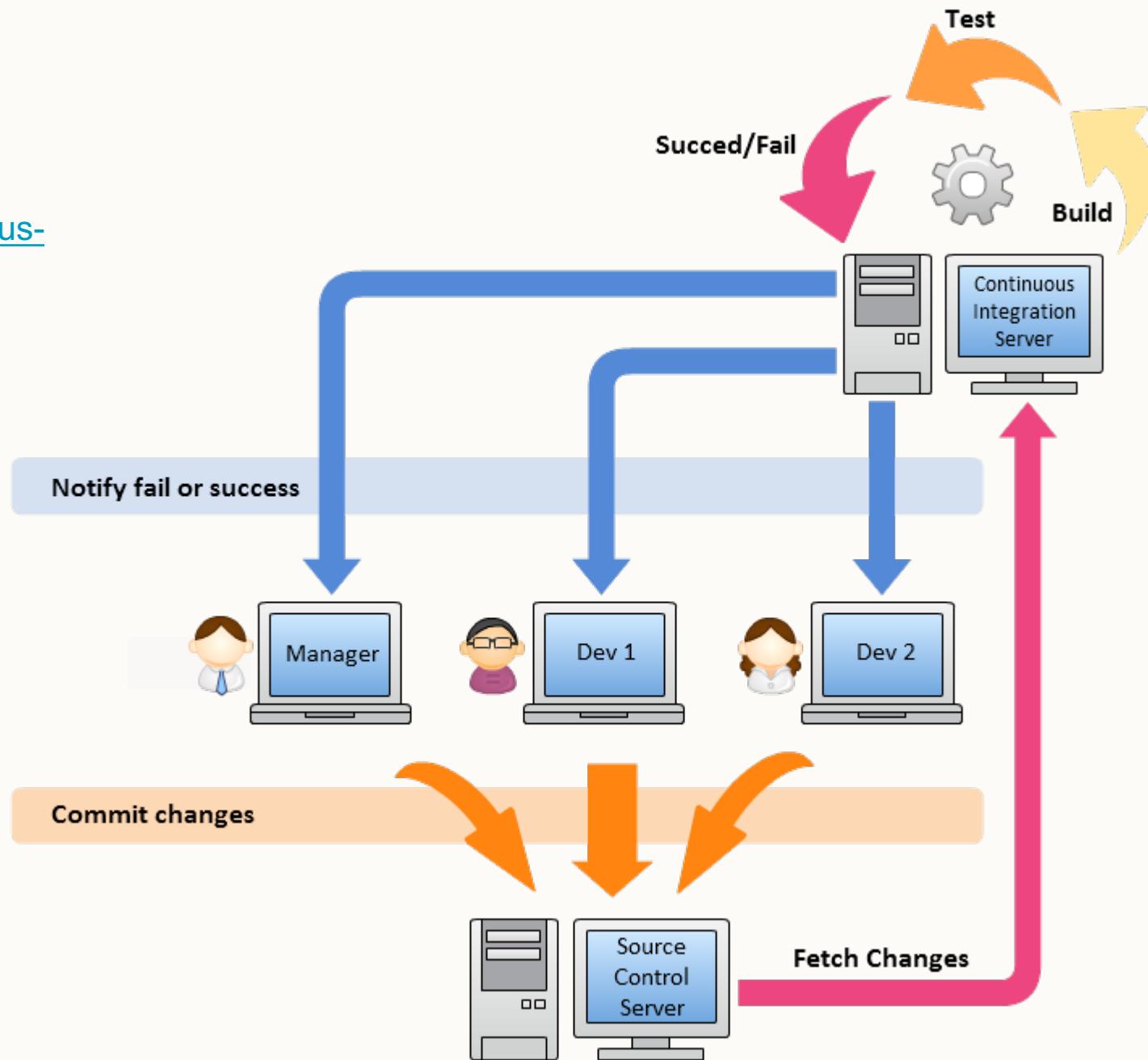
[top]

https://junit.org/junit4/faq.html#best_1

Continuous Integration Lifecycle

<https://code-maze.com/what-is-continuous-integration/>

Integrate new code often and early





Design Principles and Patterns

- From Concepts to Patterns
 - OO Concepts
 - Object model (abstraction, encapsulation, modularity, hierarchy); data abstraction; inheritance; polymorphism; interfaces
 - OO Design Principles
 - Encapsulate what varies; favour composition over inheritance (object aggregation vs class inheritance); program to interfaces, not implementations; ...
 - OO Design Patterns:
 - Show how to build systems with good OO design qualities (reusable; extensible; maintainable)



Principles and Patterns

- SOLID Principles
 - You should understand and be able to explain the idea behind the principles
- Design Patterns
 - You should understand the ones we discussed in the lectures
 - Singleton; Abstract Factory; Factory Method; Adapter; Observer; State; MVC
 - You should be able to identify core design patterns in class diagrams
 - Only the ones we have gone through
 - You don't need to be able to code them



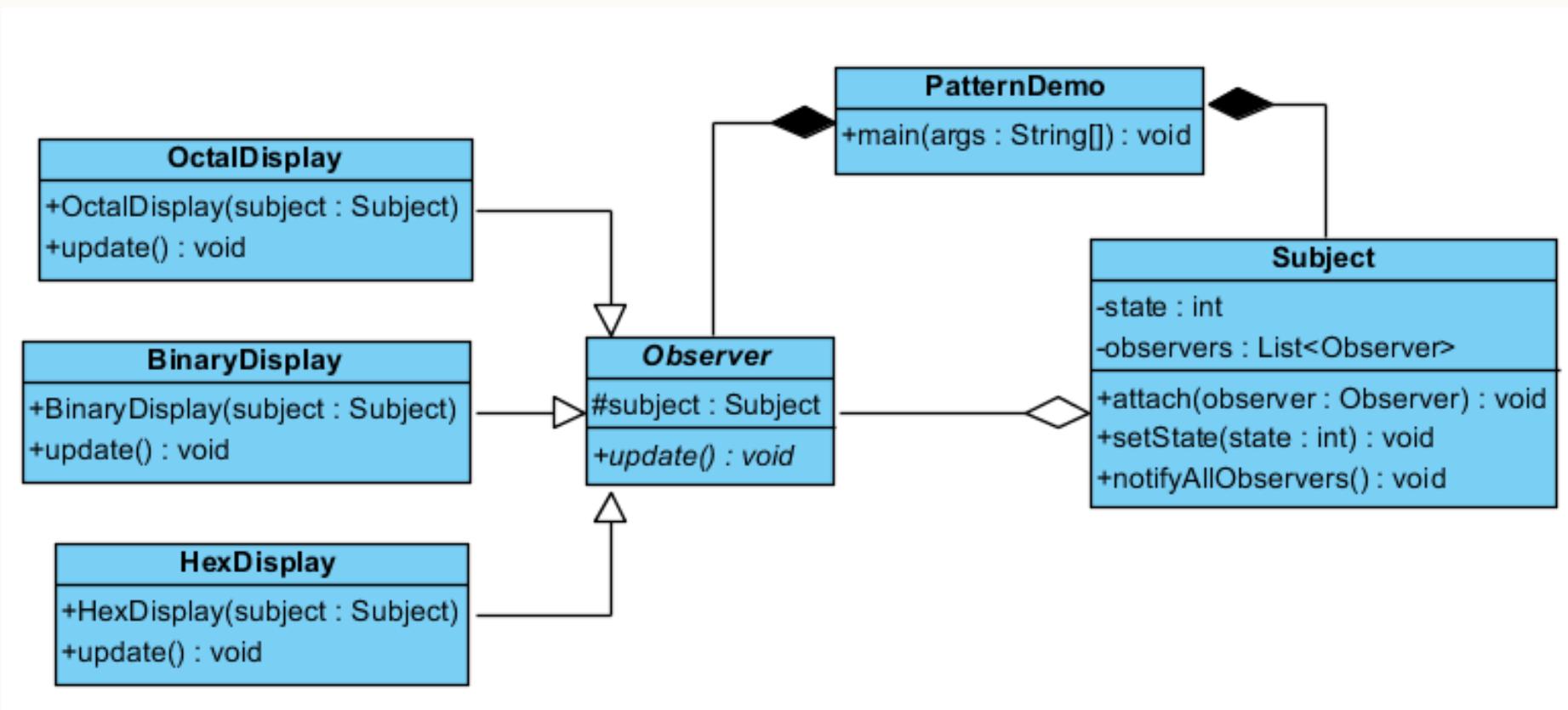
Example: Design Principle

- UB says "High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions."
- Dependency Inversion Principle
 - Goal is to reduce coupling between different pieces of code by adding a layer of abstraction



Example: Design Pattern

- What pattern? Benefits? Example for application?





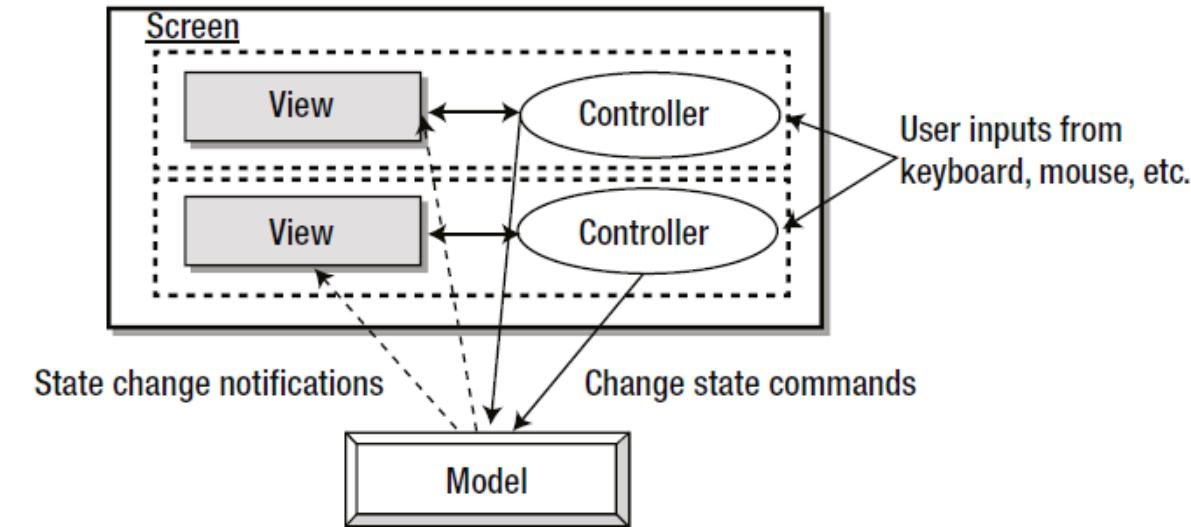
Maintainable GUI Development

- GUIs in Java
 - Swing vs. JavaFX
 - Concept of FXML and SceneBuilder
 - Event driven programming in JavaFX
 - Simple usage of CSS to style interfaces
 - MVC design pattern
- Threading principles
 - Stopping GUIs becoming unresponsive
 - The idea behind why we want to do this...



MVC Design Pattern

- MVC
 - The **model** provides a way for **views** to subscribe to its state change notifications
 - Any interested **views** subscribe to the **model** to receive state change notifications
 - The **model** notifies all **views** that had subscribed whenever a **model's** state changes





Threading

- Stopping GUIs Becoming Unresponsive
 - JavaFX launches the UI on a JavaFX Application thread
 - This thread should be left handling the UI interaction
 - Heavy computation should be done elsewhere to prevent freezing!
 - JavaFX provides a solution: The "javafx.concurrent" package
 - A way of creating and communicating with other threads with a JavaFX interface



Libraries and Communal Software Development

- Software deployment through libraries
 - Example: Java JRE > *.jar file + API + License
- Open source software development
 - Why go open source?
 - Principle of open source development
 - Open source criteria
 - Common licenses / types
 - BSD; GNU; etc.
 - Permissive; CopyLeft; etc.
 - Tracking issue and bugs
 - Mantis and Bugzilla

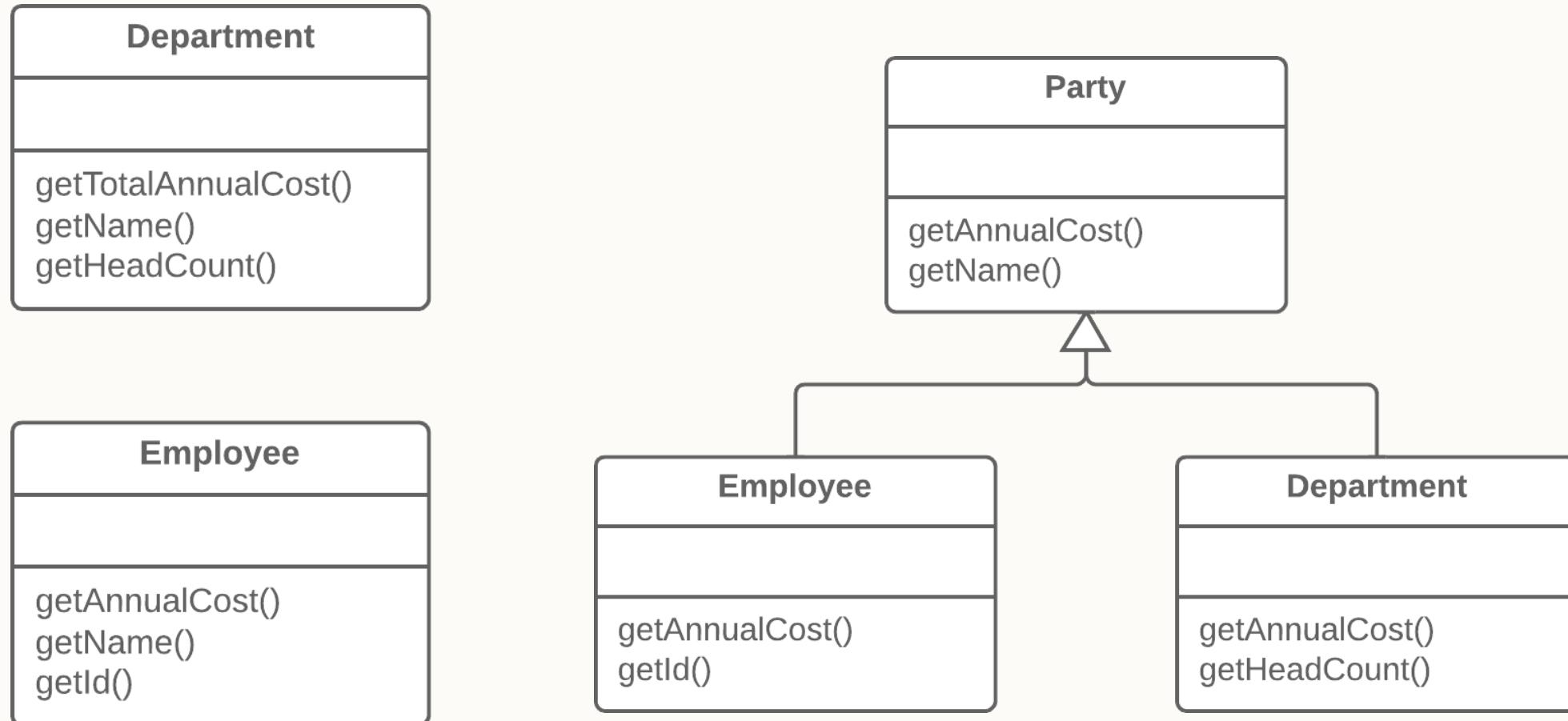


Replace type code with polymorphism

- **Switch statements are very rare in properly designed object-oriented code, much more common in procedural programs**
 - Therefore, a switch statement is a simple and easily detected “bad smell”
 - Of course, not all uses of switch are bad
 - A switch statement should *not* be used to distinguish between various kinds of object
 - Lengthy to test all of the test cases
- There are several well-defined refactorings for this case
 - The simplest is the creation of subclasses



Calculating annual cost examples





Exam Formalities



Format of Exam

- The exam (worth 25%) will be using the UoN Rogo system
 - Check your exam timetable to find out which room you have to go to!
- Mock exam (will be announced)
 - Get an idea of the types of questions you are going to get in the actual exam
- Different question styles (multiple choice; text answers; diagram labelling; etc.)
 - You will not be asked to write big code sections from scratch but you will be asked questions about code and perhaps to improve given code



Time and Location of the Exam

- Time and location
 - Time
 - Friday XX/XX/2024; Xam
 - Location
 - No room yet
 - Will be multiple rooms
 - **Make sure you go to the correct one!**





Rogo Mock Exam

High level Feedback



Rogo Mock Exam

- Real exam
 - 100 marks of material to be completed in 60 minutes
- Mock exam
 - 35 marks of material to be completed in 21 minutes
- Exam will be on multiple; you can always go back to previous questions
- Some parts of the exam will be automatically marked (e.g. multiple choice)
- Some parts of exam will be manually marked (e.g. text)





Rogo Mock Exam to Try

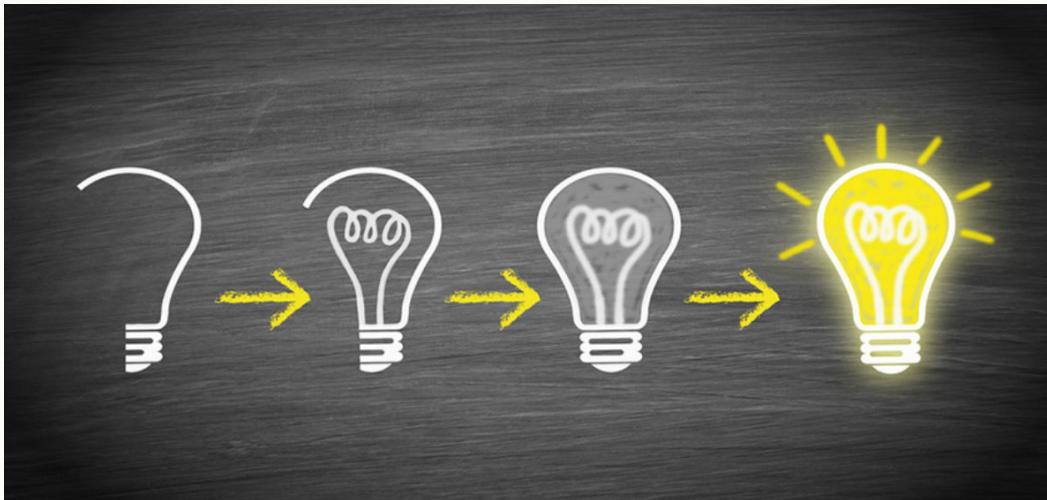
- Mock exam
 - As well as seeing the kind of questions you will encounter, it is also a chance to experience the Rogo online system which is what you will use in the exam.
 - The system will give you feedback, but obviously some of the questions need manual marking, so you will have to look at the answer and judge for yourself if you have scored.





And Finally ...

- What have you learned in the module and how does it help you in the future?



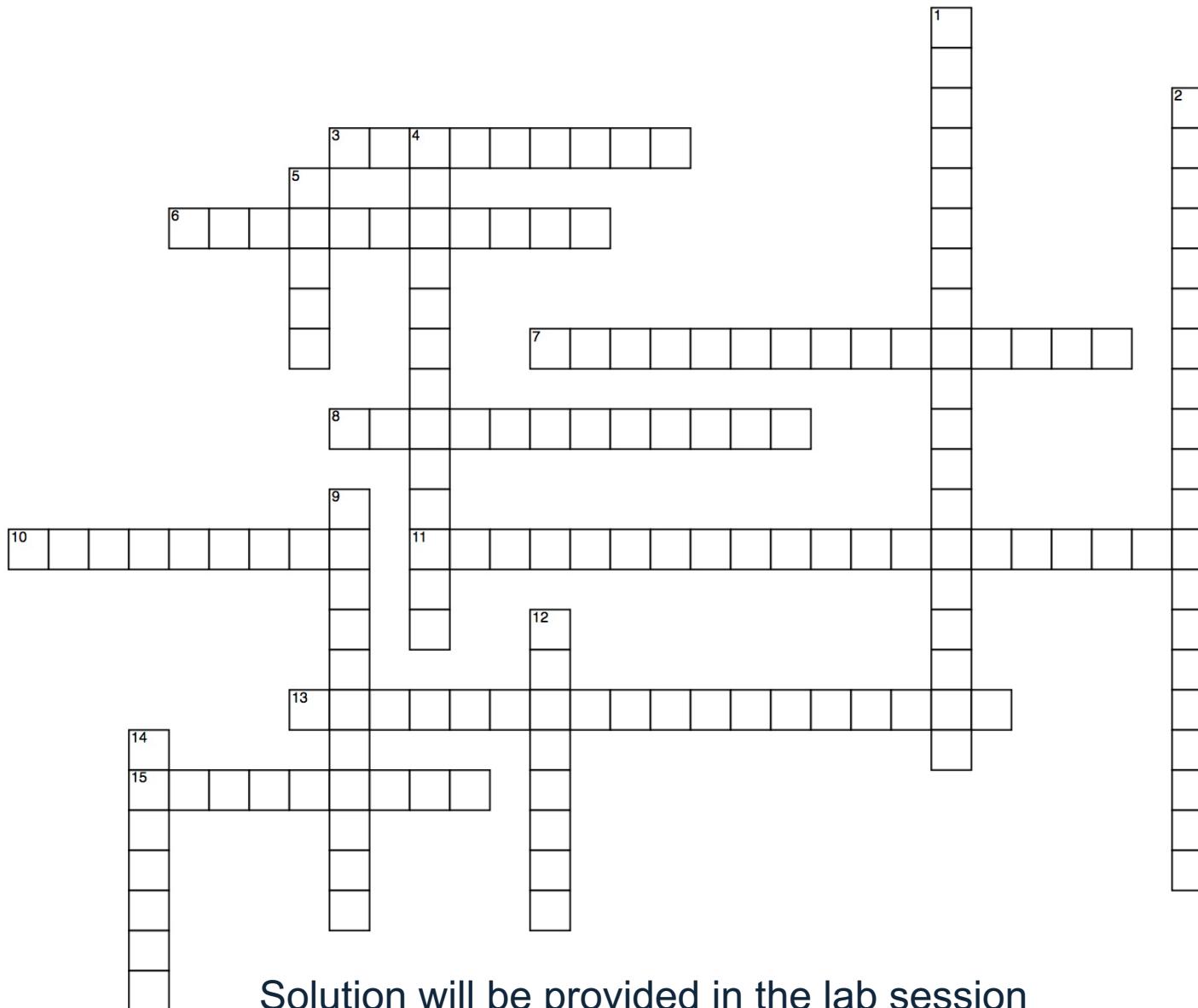
- And the last word for your revision:
 - Lecture recordings are not the most reliable source of information
 - You should use them only to complement other more reliable resources



Crossword Puzzle

SWM Design Principles and Patterns

Object orientation, SOLID and GOF Patterns Puzzle



ACROSS

- 3 Describes the use of plugins and APIs
- 6 Expose essential features while hiding irrelevant detail
- 7 Uninstantiated creational pattern
- 8 Adaptation to a specific usage
- 10 For creating one and only one instance
- 11 Clients should not be forced to depend on methods that they themselves do not use
- 13 If it quacks like a duck and swims like a duck but needs batteries you are probably breaking this principle
- 15 Flexible alternative to using inheritance

DOWN

- 1 High and low level modules should depend on abstractions
- 2 One class should do one thing
- 4 To modularise class functionality
- 5 To allow object behaviour to change at run time
- 9 Passing on behaviours from super classes to subclasses
- 12 One-to-many behavioural pattern
- 14 Is a wrapper for interfaces