COMP2059 Developing Maintainable Software

LECTURE 01 – THE CHALLENGES OF SOFTWARE MAINTENANCE

Boon Giin Lee (Bryan)

Welcome to AES DMS

Lecture

Dr. Boon Giin Lee (Bryan)

Office: PMB424

Email: boon-giin.lee@nottingham.edu.cn

Consultation Hours: Thursday (13:00 to 15:00)

Computing Lab

Dr. Heng Yu

Office: PMB426

Email: heng.yu@nottingham.edu.cn

Consultation Hours: Wednesday (13:00 to 16:00)

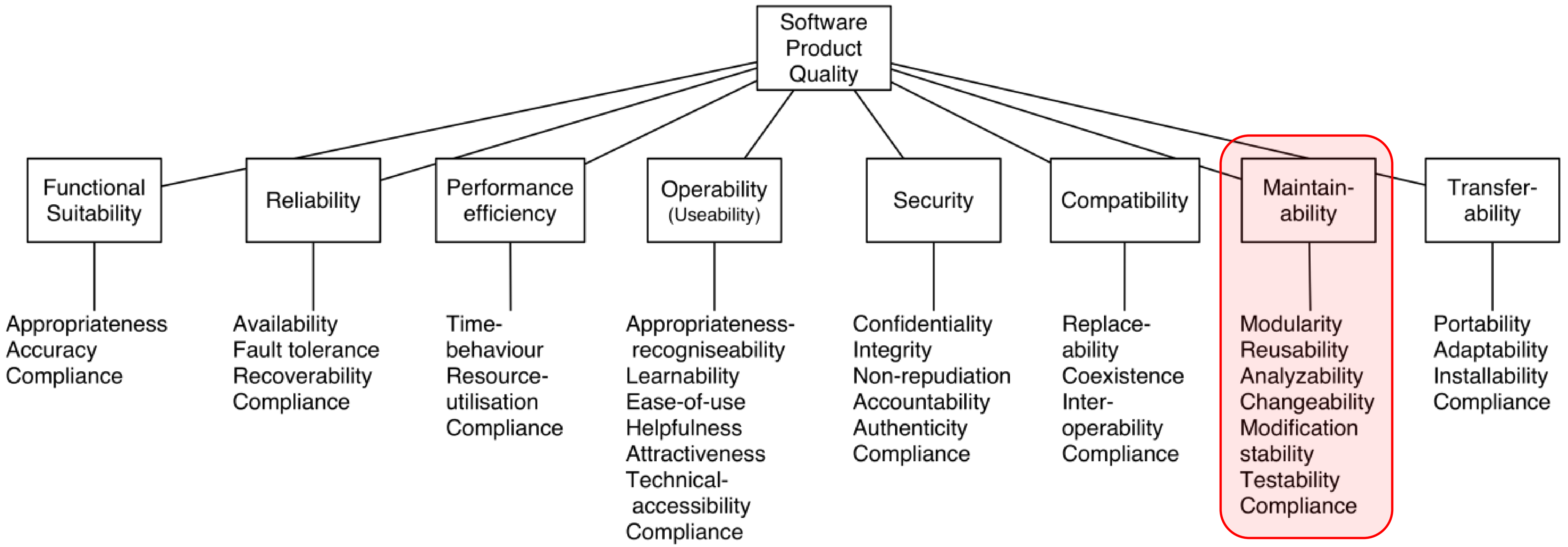
Software Maintenance

- What is it?
  - "Modification of software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment." --- IEEE Standard for Software Maintenance (https://ieeexplore.ieee.org/document/257623)

- Also need to consider how to reduce the effort of maintenance!
  - Building software that is easy to maintain and extend in the first place!

Maintainable Software

- ISO 25010: Software Product Quality


Software Product Quality
Functional Suitability
Reliability
Performance efficiency
Operability (Useability)
Security
Compatibility
Maintain-ability
Transfer-ability
Appropriateness Accuracy Compliance
Availability Fault tolerance Recoverability Compliance
Time-behaviour Resource-utilisation Compliance
Appropriateness-recogniseability Learnability Ease-of-use Helpfulness Attractiveness Technical-accessibility Compliance
Confidentiality Integrity Non-repudiation Accountability Authenticity Compliance
Replace-ability Coexistence Inter-operability Compliance
Modularity Reusability Analyzability Changeability Modification stability Testability Compliance
Portability Adaptability Installability Compliance

University of Nottingham
UK | CHINA | MALAYSIA

Maintainable Software

- Modularity – can we divide the code in modular form?
- Reusability – can we reuse the same module/code?
- Analyzability – is it easy to analyze the code?
- Changeability – is it easy to change the code?
- Modification Stability – will it affect other code?
- Testability – how can we test the code?
- Compliance – does the code works in different OS or version?

Maintainable Software

- Three **principles** for building maintainable software:
  - Maintainability benefits most from adhering to simple guidelines.
  - 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.

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.

The Importance of Knowing How to Do Software Maintenance Efficiently

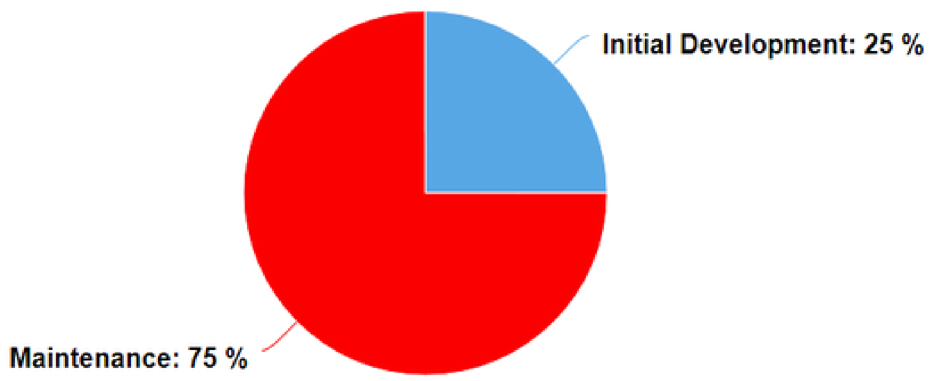
Resources on Initial Development vs Maintenance

- It’s important to learn about maintenance.
- It’s important to build maintainable software.

	Resources spend	
Author	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/

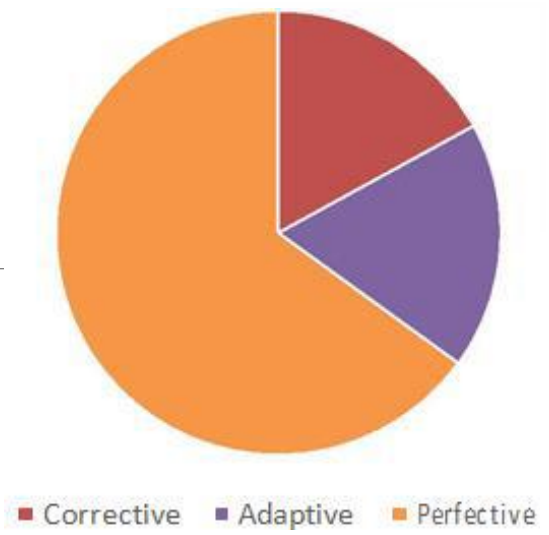
Resources spent on initial development vs. maintenance



https://www.coretechnologies.com/blog/alwaysup/vs-custom-windows-service/

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

- Corrective Maintenance
  - Finding and fixing errors in the system.
    - e.g., bugs.
- Adaptive Maintenance
  - The system must 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.

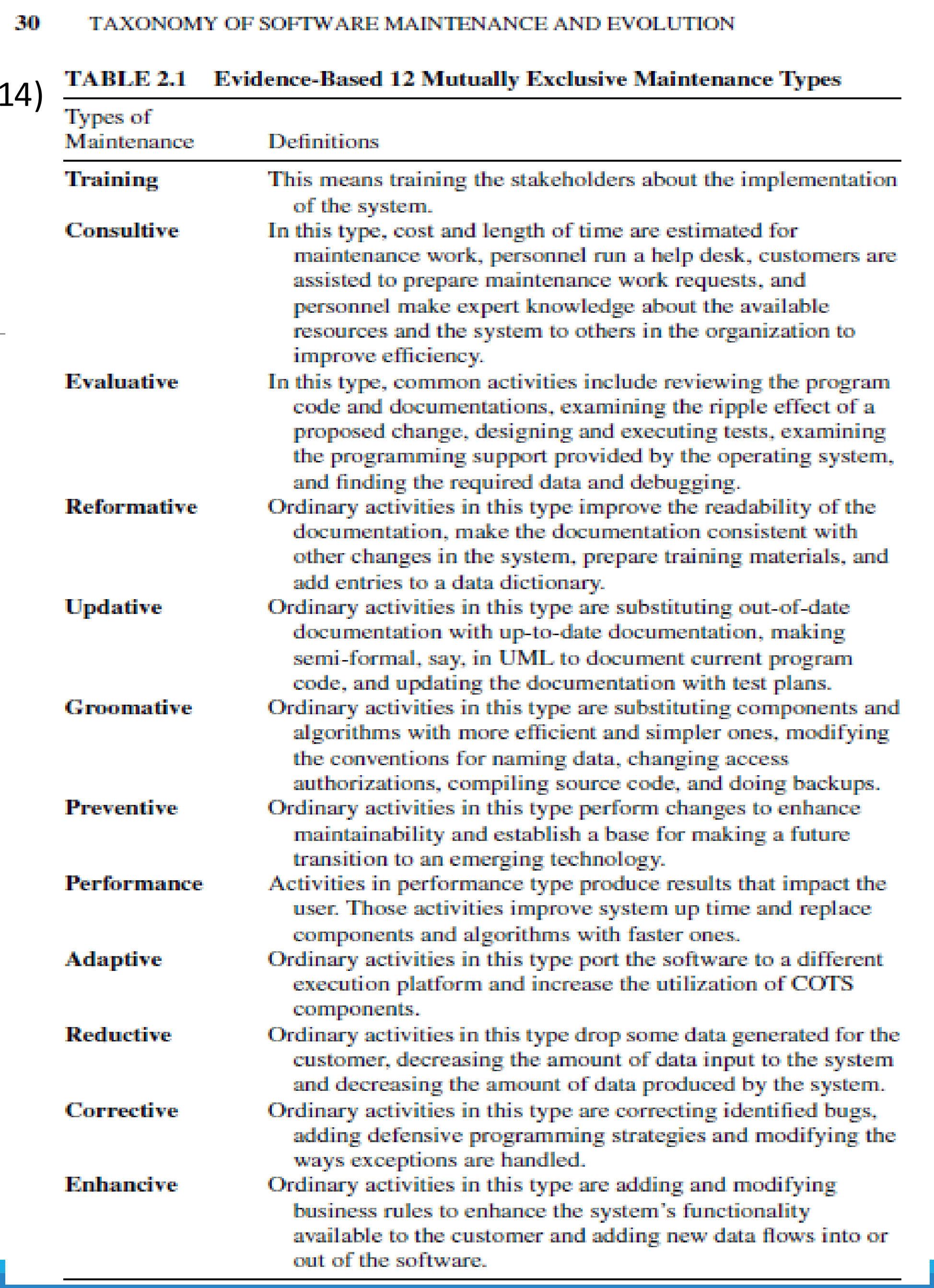


University of Nottingham
UK | CHINA | MALAYSIA

Tripathy and Naik (2014)

A Lots More!

- Maintenance vs Evolution?
  - Maintenance:
    - Preserving software in a working state.
  - Evolution:
    - Improving software.
- What will be learn will largely be applicable to both.



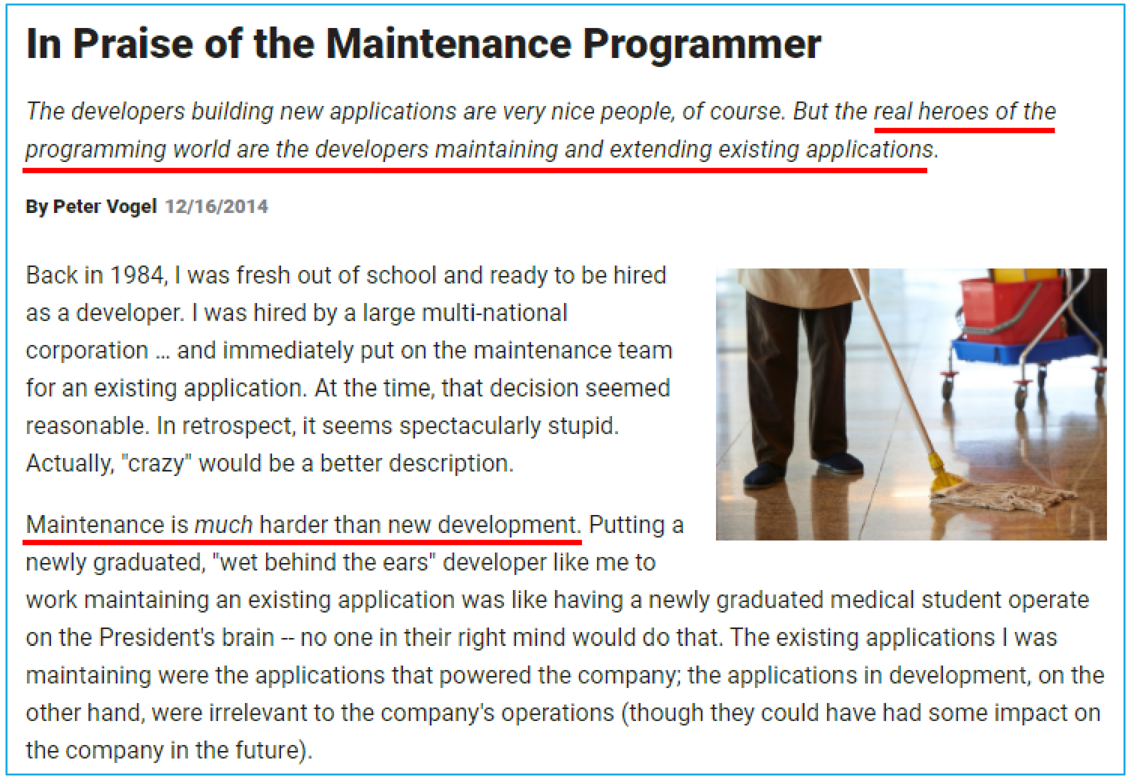
30 TAXONOMY OF SOFTWARE MAINTENANCE AND EVOLUTION

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.

Building vs Maintaining Software

- This is harder than writing new software!



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. Putting a newly graduated, "wet behind the ears" developer like me to work maintaining an existing application was like having a newly graduated medical student operate on the President's brain -- no one in their right mind would do that. The existing applications I was maintaining were the applications that powered the company; the applications in development, on the other hand, were irrelevant to the company's operations (though they could have had some impact on the company in the future).

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

other day, 'How often do you have to
update your libraries to make sure

https://www.youtube.com/watch?v=Za27pGeg--s

What is Involved in Software Maintenance?

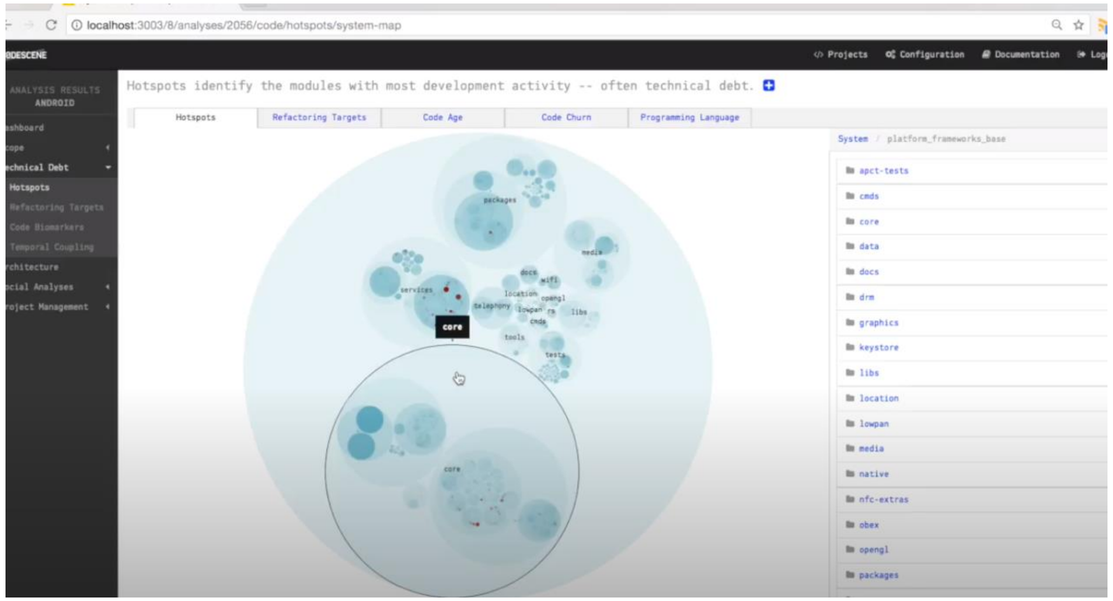
- Understanding the client.
- Understanding the code.
- Refactoring the code.
- Extending the code.
- Working as a team.
- Managing client expectations.
- Managing maintenance process.

Maintenance as a Murder Mystery

- Fixing a problem requires a 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/)
    - Identify patterns in the evolution of your code.
    - Software forensics


localhost:3003/8/analyses/2056/code/hotspots/system-map

Hotspots identify the modules with most development activity -- often technical debt.

Watch first 3 minutes
https://www.youtube.com/watch?v=n4P_I9rXKbE

Why COMP2059?

I HAVE LEARNED EVERYTHING IN QY!

Previous Topics You Have Covered

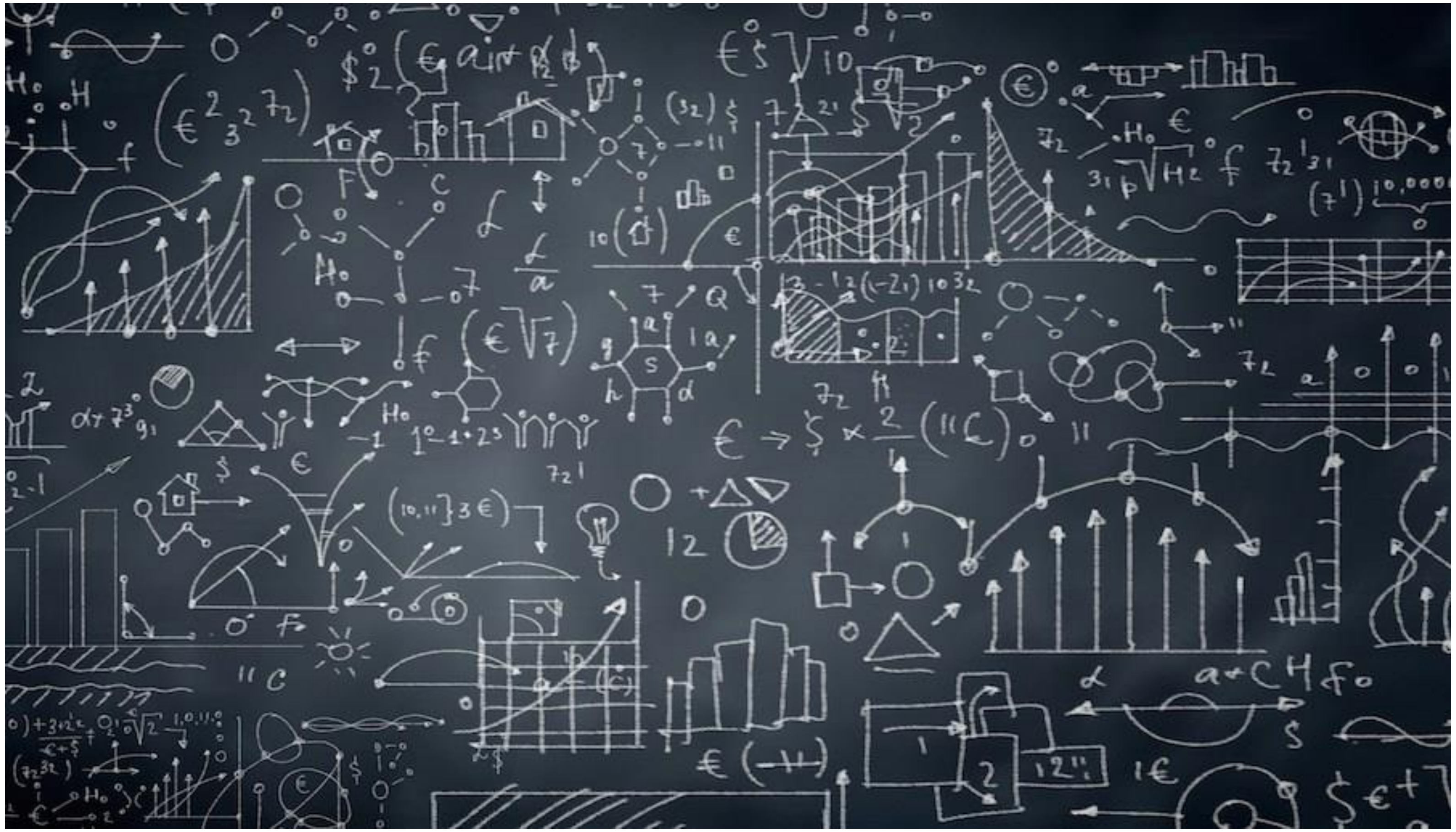
- Programming in Java
  - Basics
  - Arrays
  - Classes and Objects
  - Interfaces
  - Exceptions
  - Strings
  - Inheritance
  - Java Collections
  - Software Testing
  - Refactoring

Previous Topics You Have Covered

- Software Engineering

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

Imagine Looking at Someone Else Code

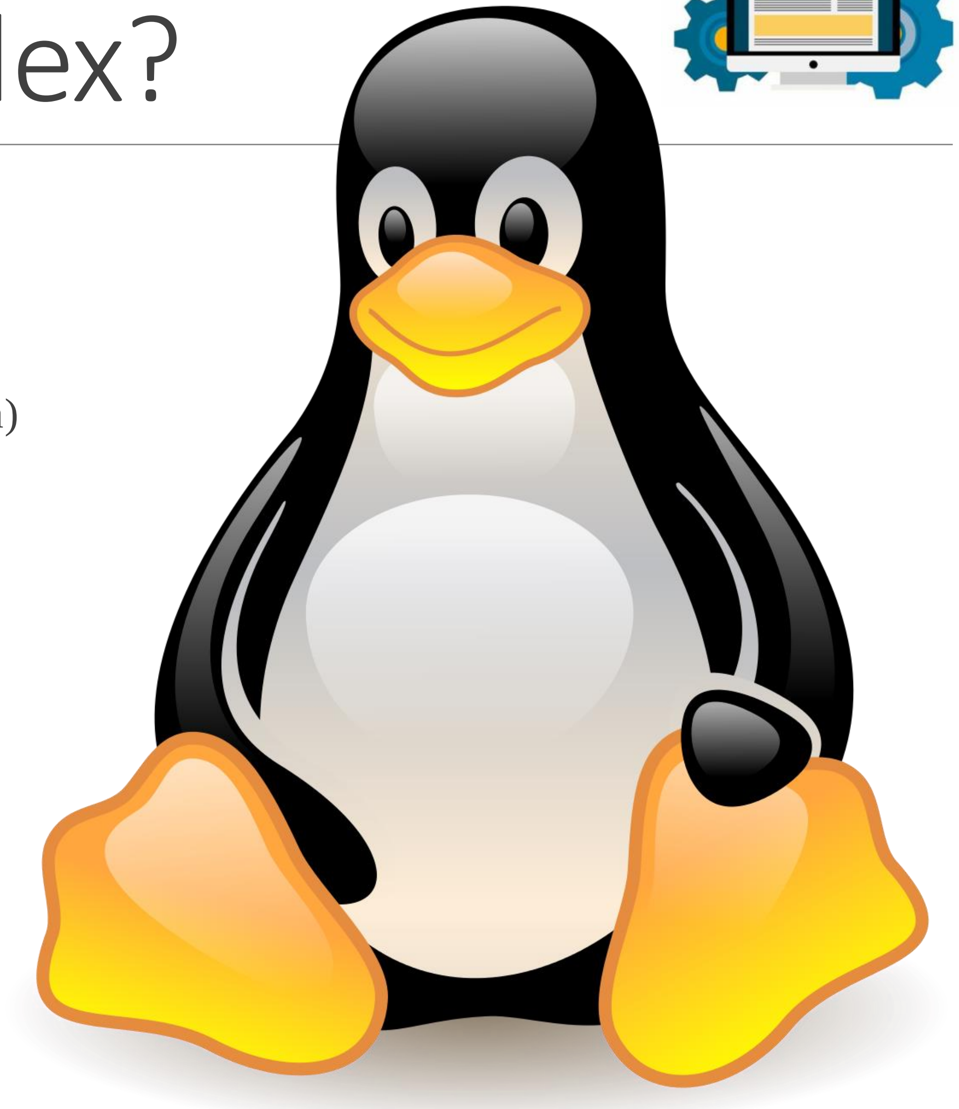


It’s All About Big Software Projects

- It’s about …
  - How to write them from scratch so you can maintain them more easily later.
  - How to approach an existing mass/mess of code!

- All this is very practical advice that you will find useful in real world.
  - In short term, it should give you lots of pointers for GRP.

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?
    - Around 10,000 patches in each release
    - Releases every 2 to 3 months
  - How many developers?
    - Each release by 1,000+ of developers
    - And 200 corporations

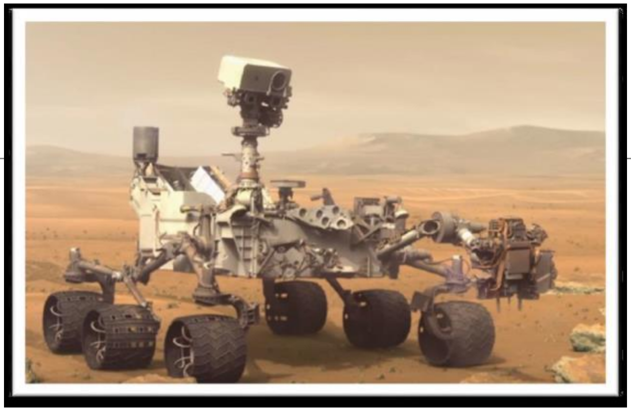
How Big is Big? How Complex is Complex?

- Example: Windows 11

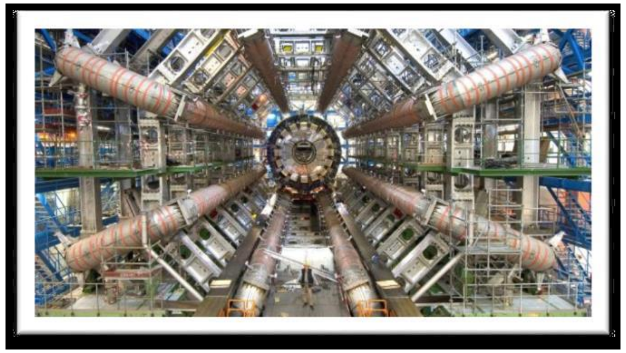
  - How many lines of code?
    - Estimated 60 to 100 million lines of code.
    - Include Windows Kernel, networking components, drivers, and other applications.
    - Primary written in the C++, C# (some components) and others.
    - (Mac OS X is estimated to be between 40 to 50 million lines of code.)

  - How many developers?
    - Estimated around 8,000 people involved, including writers, testers and designers.

How Big is Big?



- Curiosity rover: 2.5 million lines.



- CREN: 3 million lines (+50 million in projects).
  - Large hadron collider (particle accelerator).
- The code that guided the Apollo 11 module in 1969: AGC ~ 100,000 lines.

How COMP2059 Builds on Previous Work?

- What have seen in previous lectures is how to develop toy examples.

- Real world software is a different beast.
  - Messy design.
  - Complex.
  - Multiple authors.
  - Bad coding and bad commenting.
  - Lots of dependencies.

- Often you cannot familiarise yourself with the complete code base.

An Analogy

- QY
  - T h i s (terminology)
  - This (illustrative examples)
  - This a ful sentence is? (some OOP and SE to put it together)
- P1
  - This is a full sentence.
  - This is a full sentence, as it has a subject, verb and object ☺

The Overall Learning Objectives

- To ensure you understand that **software maintenance is central** to software development.
  - Maintainable software and software maintenance are 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 development ‘butterfly’.

Challenges of Developing Maintainable Software

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?

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.

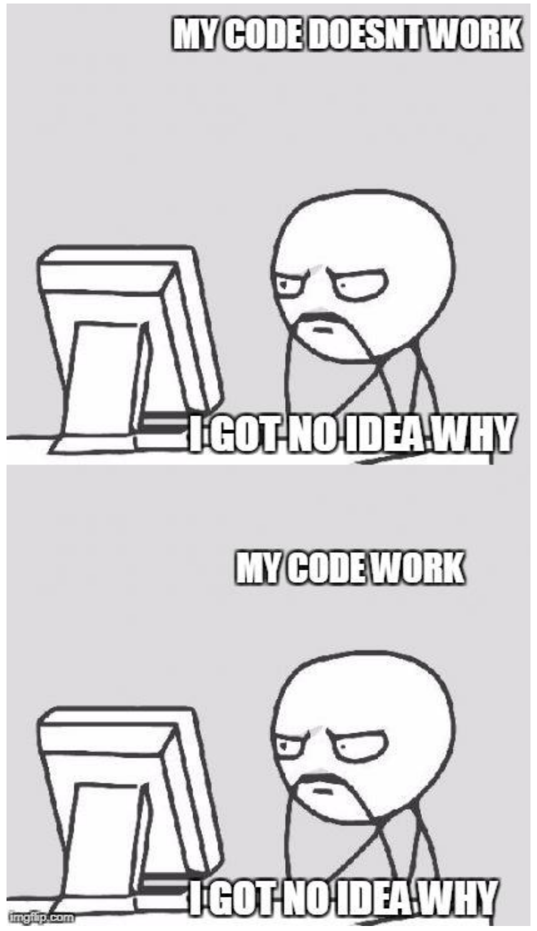
University of Nottingham
UK | CHINA | MALAYSIA

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).

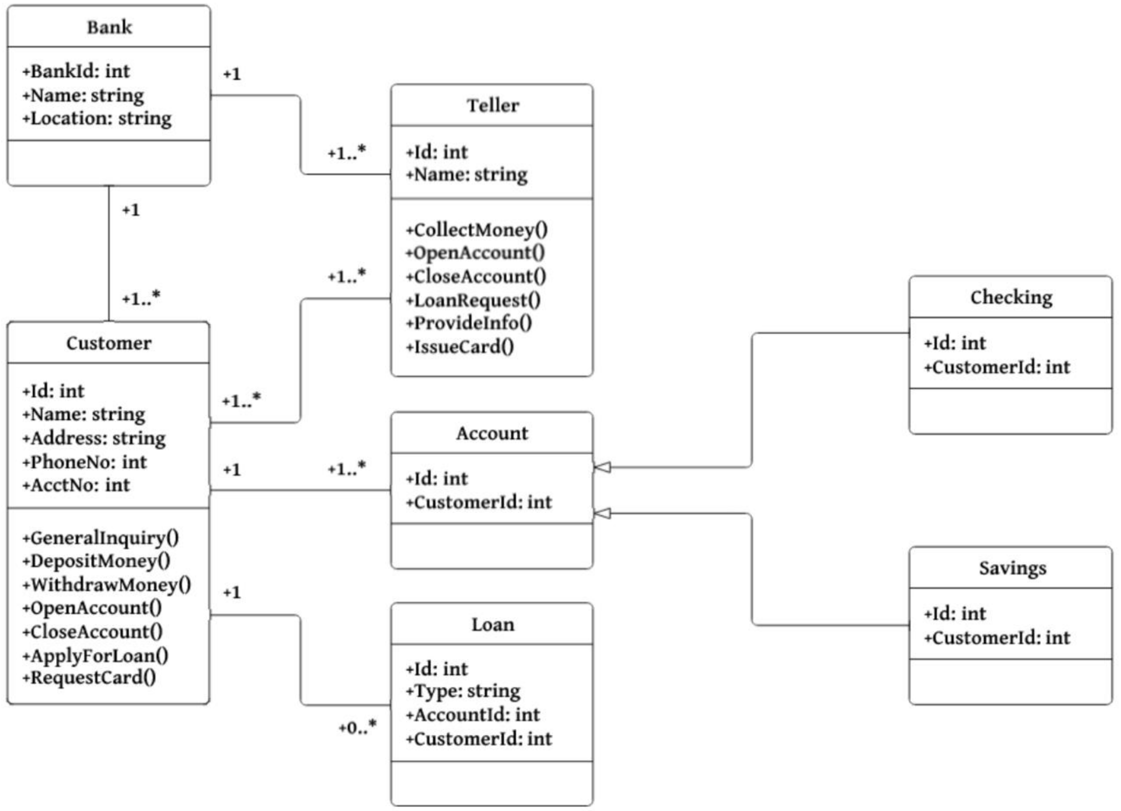
1a – Understanding the Code

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



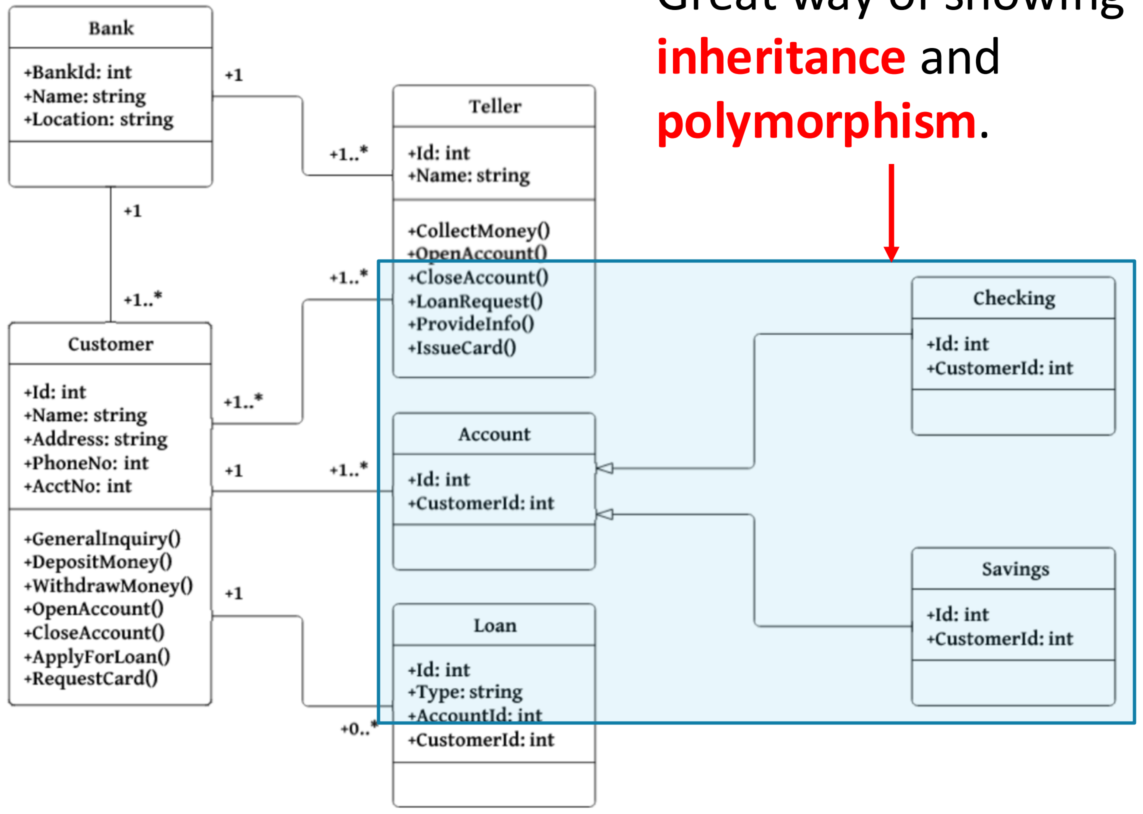
1a – Making Sense of System Structure with Class Diagrams

- Class diagrams
  - Show a set of classes and their relationship.
  - Addresses static design view of a system.
  - Classes.
    - Blueprints (templates) for objects.
    - Contains data/information and perform operations.



1a – Making Sense of System Structure with Class Diagrams

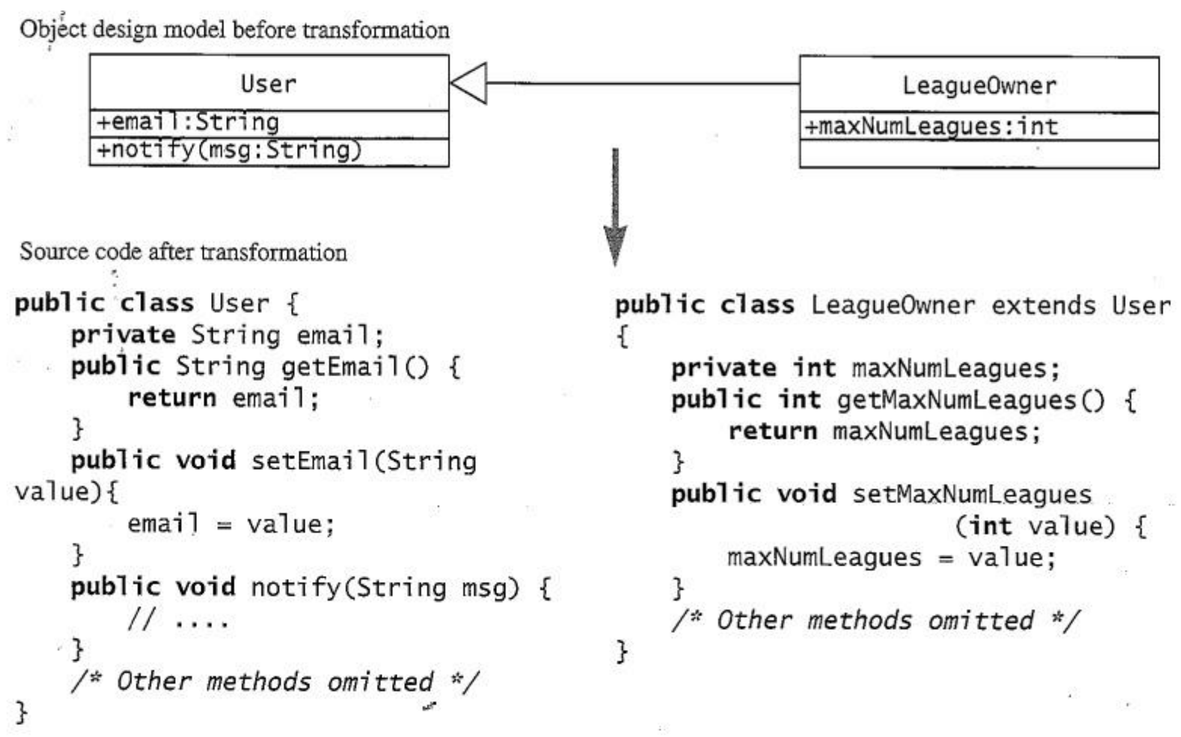
- Class diagrams
  - Show a set of classes and their relationship.
  - Addresses static design view of a system.
  - Classes.
    - Blueprints (templates) for objects.
    - Contains data/information and perform operations.



Great way of showing **inheritance** and **polymorphism**.

1a – Reverse Engineering

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


Object design model before transformation

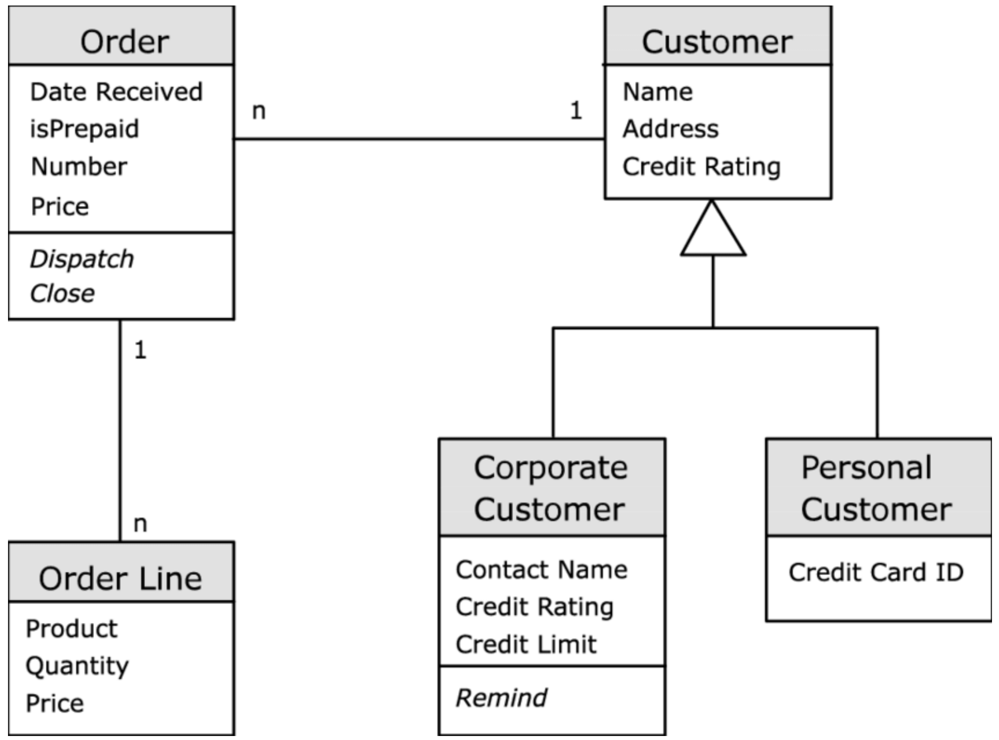
Source code after transformation

```
public class User {
    private String email;
    public String getEmail() {
        return email;
    }
    public void setEmail(String
value){
        email = value;
    }
    public void notify(String msg) {
        // ....
    }
    /* Other methods omitted */
}
```

```
public class LeagueOwner extends User
{
    private int maxNumLeagues;
    public int getMaxNumLeagues() {
        return maxNumLeagues;
    }
    public void setMaxNumLeagues
                        (int value) {
        maxNumLeagues = value;
    }
    /* Other methods omitted */
}
```

1a – Reverse Engineering

- Simple class diagram (only including relevant methods).

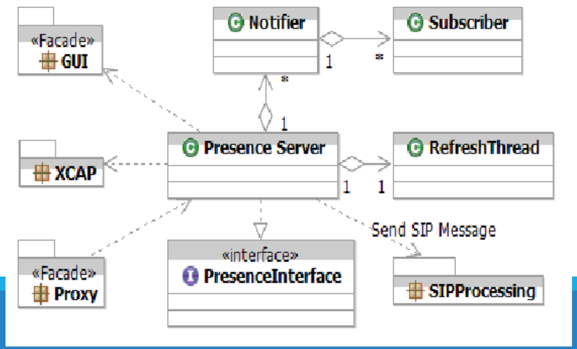


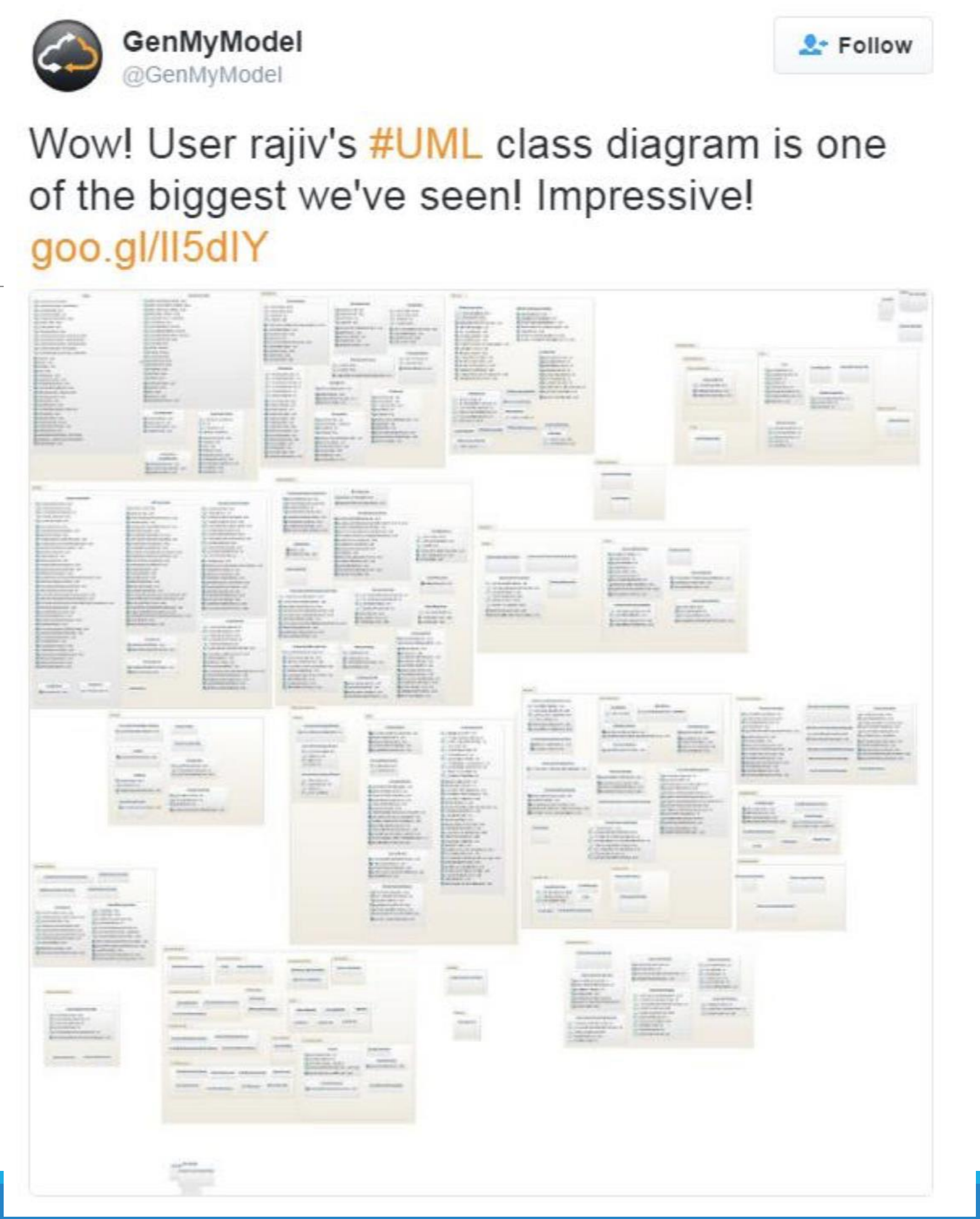
1a – UML in Practice

- Class diagrams can help sometimes (but not always).
- They summarise the content of classes and relationships between them in the simplest way possible.
- Does it make sense to build a *class diagram* for an entire project?
  - There are other methods to do this and other UML techniques to look at software at a higher level (e.g., deployment diagrams).
- Not forgetting, code can be generated directly from UML for maximum consistency – why is this a good thing?
  - https://www.visual-paradigm.com/support/documents/vpuserguide/276/381/7486_generateorup.html (Not available in Community Edition)

1a – UML in Practice

- Caution! This is 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.

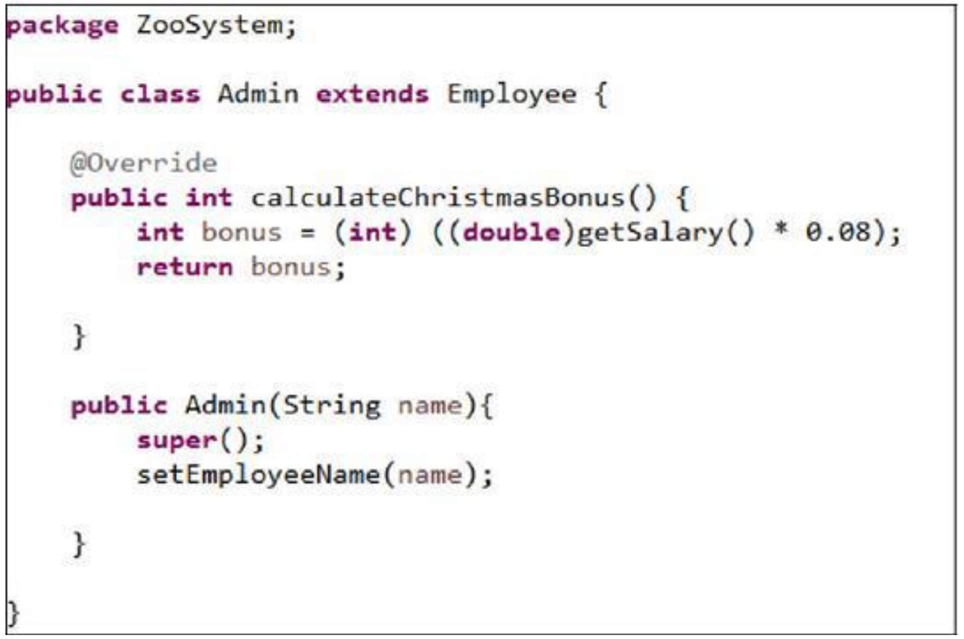



GenMyModel @GenMyModel
Follow

Wow! User rajiv's #UML class diagram is one of the biggest we've seen! Impressive! goo.gl/Il5dlY

1b – Understanding the Code Itself

- As well as understanding the structure, we need to understand what the code does.
- Task: add a method to set an employee phone number.
- You need to understand the code itself to make the right decisions for producing robust and maintainable code.

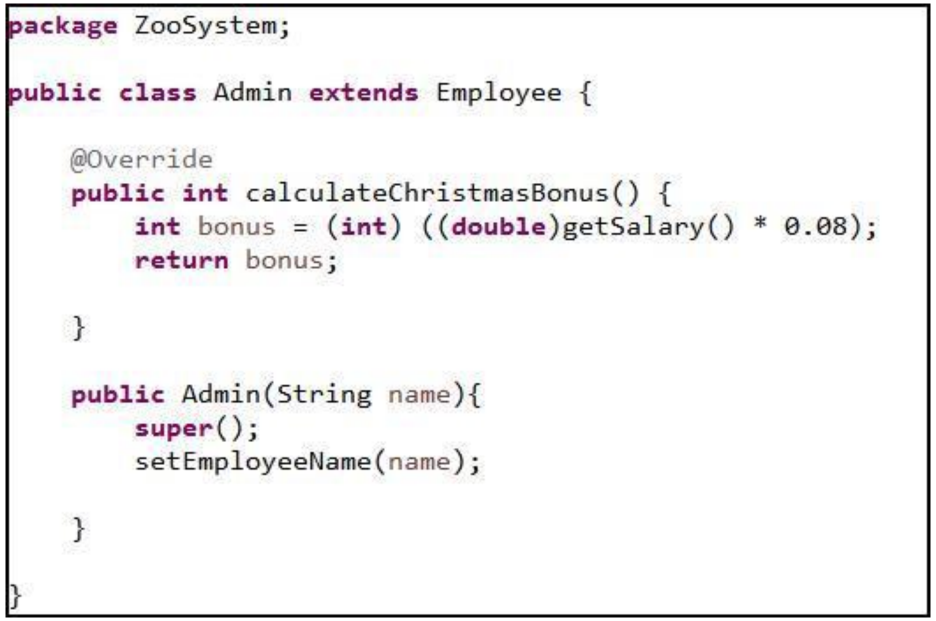


```
package ZooSystem;

public class Admin extends Employee {

    @Override
    public int calculateChristmasBonus() {
        int bonus = (int) ((double)getSalary() * 0.08);
        return bonus;

    }

    public Admin(String name){
        super();
        setEmployeeName(name);

    }
}
```

University of Nottingham
UK | CHINA | MALAYSIA

1b – Understanding the Code Itself

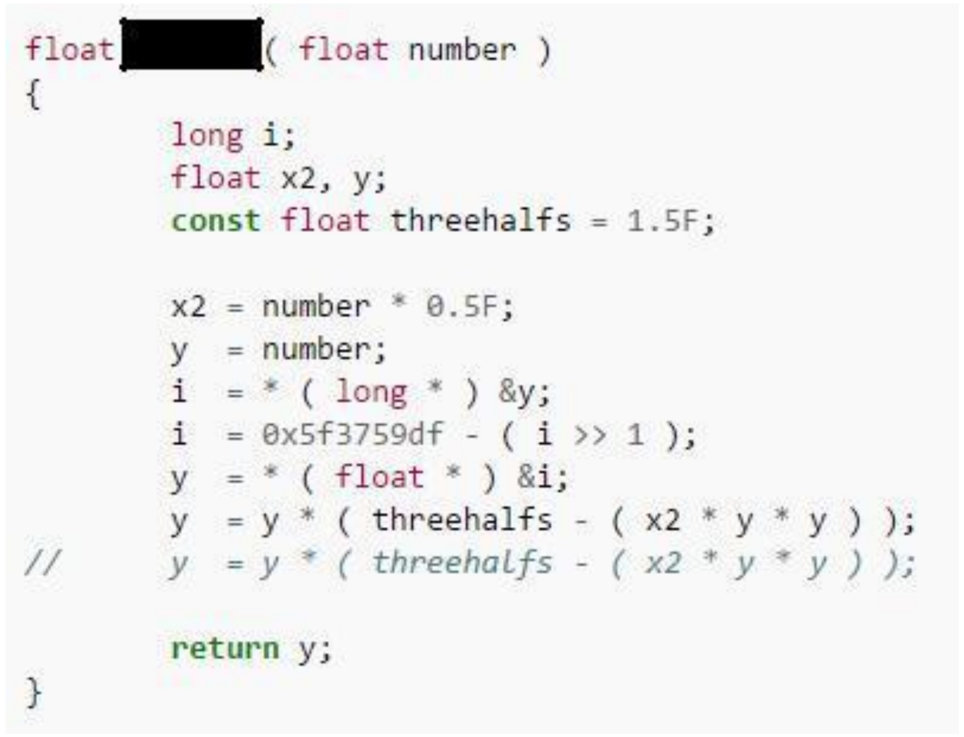
- Which class should it go in?
  - In “Admin”, or the abstract class “Employee”?
- Should it be public or private?
- What variable will you use?
  - What type?
- What if another subtype of “Employee” already has a telephone method?
  - And what if one doesn’t have a fixed telephone? (e.g., cleaning staff?)



```
package ZooSystem;

public class Admin extends Employee {

    @Override
    public int calculateChristmasBonus() {
        int bonus = (int) ((double)getSalary() * 0.08);
        return bonus;

    }

    public Admin(String name){
        super();
        setEmployeeName(name);

    }
}
```

1b – Following Conventions vs Being Smart

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



```
float [redacted]( 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;
}
```

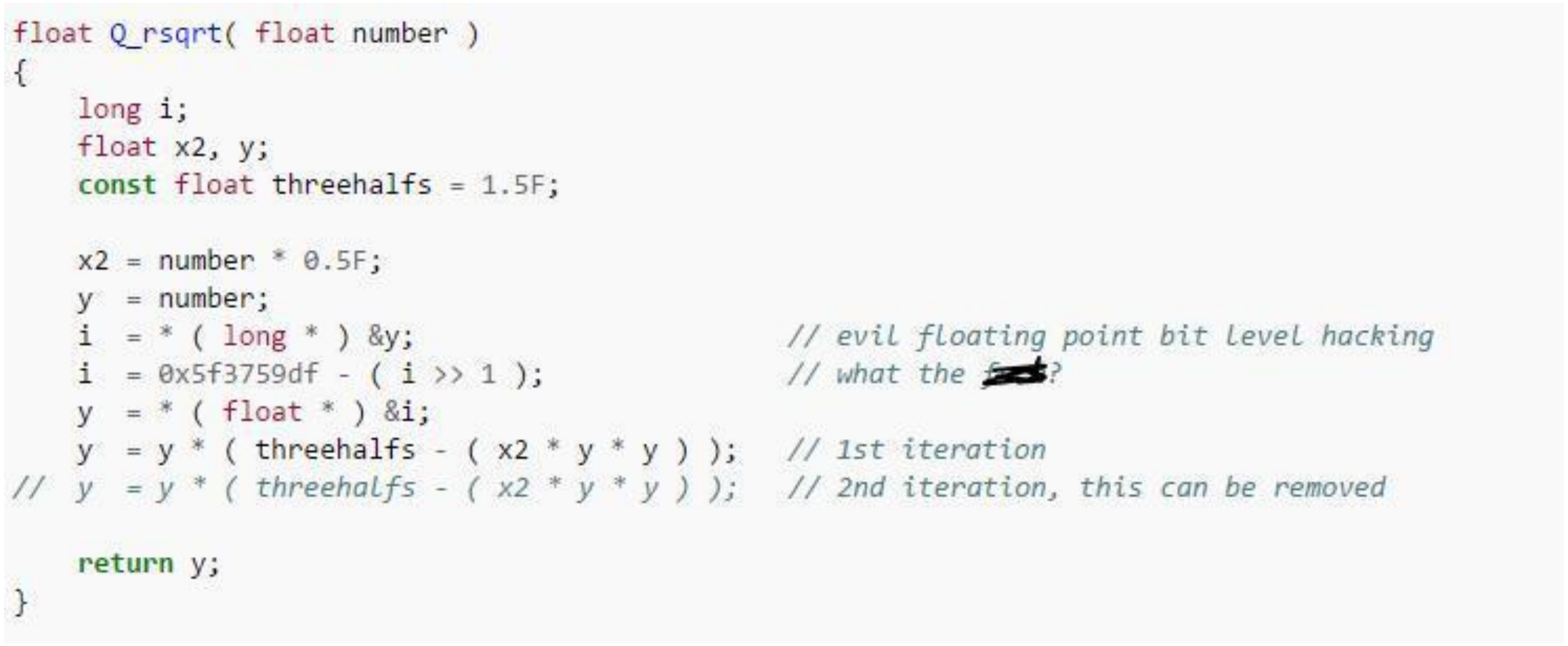
- No comments.
- Poor variable names.
- Obscure coding.
- Redundant commenting out.

Actually (yes, really) Quake3's Fast InvSqrt()

Tonight, get out your last piece of Java coursework from QY, and see it you still understand what you did 6 months ago?

1b – Following Conventions vs Being Smart

- 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 [redacted]?
    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;
}
```

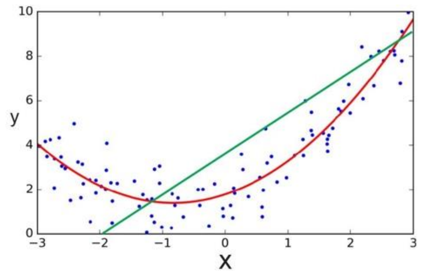
2 – Extending Existing Code

- Realistically, developers rarely start with an empty project.

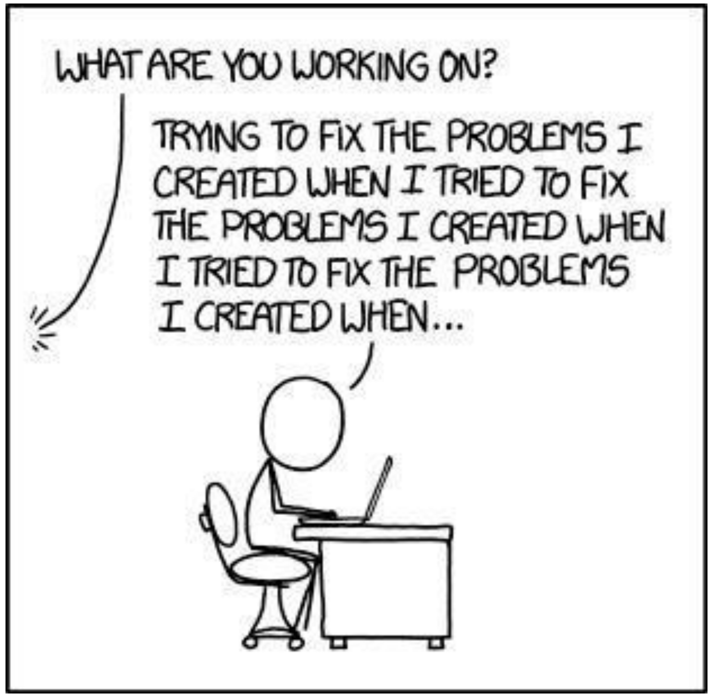
- Most projects are evolutions or enhancements to codebases or utilise existing APIs, achieved through understanding these elements of good practice.
  - Object orientation, object orientation and more object orientation.
  - Coding tools and techniques, including IDEs, Git and UML.
  - How to write consistently good code in the first place, to make extensions later easier to do.
  - How to use design patterns and SOLID principles.
  - How different libraries can interface existing code
    - e.g., creating GUIs on a pre-existing backend codebase.

2 – How Do You Know You Haven't Broken It?

- When you change existing code, you risk breaking it. To check for this, 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.



WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

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 module, 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 it someone else will need it.
- You have looked at Git and version control in the QY, we will be going over it again, and again, … , until you use it by habit.
- Understanding how open source and software licensing works.

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 words.
  - Team meetings and a design process – e.g., Scrum,
  - Team communication,
  - Source code management (e.g., Git),
  - Server-side testing (continuous integration server).

- P.S. this means talking to other people.

Things We Already Know

- You have had Java overview lectures from Pushpendu and Tony.
- FSE (COMP1035) has introduced Agile to you.
- Tony has enticed you with his Haskell.
- Some of you are very accomplished programmers.
- You will have seen some of the contents before.
- “This module is dry”.

- It’s not. It’s practical. It’s relevant. It will get you a job!

Learning Outcomes

- Software maintenance is essential.
- Writing maintainable code is awesome.
- Important concepts:
  - Many aspects of maintenance.
  - Good maintenance starts with good design.
  - Object-oriented programming (OOP).
  - Ways to visualise code (UML).
  - Managing maintenance.
- Overview of the module – make sure you note down the key dates!

References

- Visser *et al.* (2016), Building Maintainable Software – Java Edition.
- Tripathy and Naik (2015), Software Evolution and Maintenance.
- Good Developers vs Bad Developers:
  - https://medium.com/@CodementorIO/good-developers-vs-bad-developers-fe9d2d6b582b
- Kernighan & Pike, Practice of Programming Chapter 1 (Style).
- Introduction to OMGs Guide to UML.
  - https://www.uml.org/what-is-uml.htm

IDEs + Java Releases

University of Nottingham
UK | CHINA | MALAYSIA

Useful Website to Learn More About IDEs

- Eclipse + Java
  - https://www.tutorialspoint.com/eclipse/index.htm
- IntelliJ
  - https://www.tutorialspoint.com/intellij_idea/intellij_idea_getting_familiar.htm
- JetBrains: IntelliJ Video Tutorials
  - https://www.tutorialspoint.com/intellij_idea/index.htm

What’s New in Java (highlights)

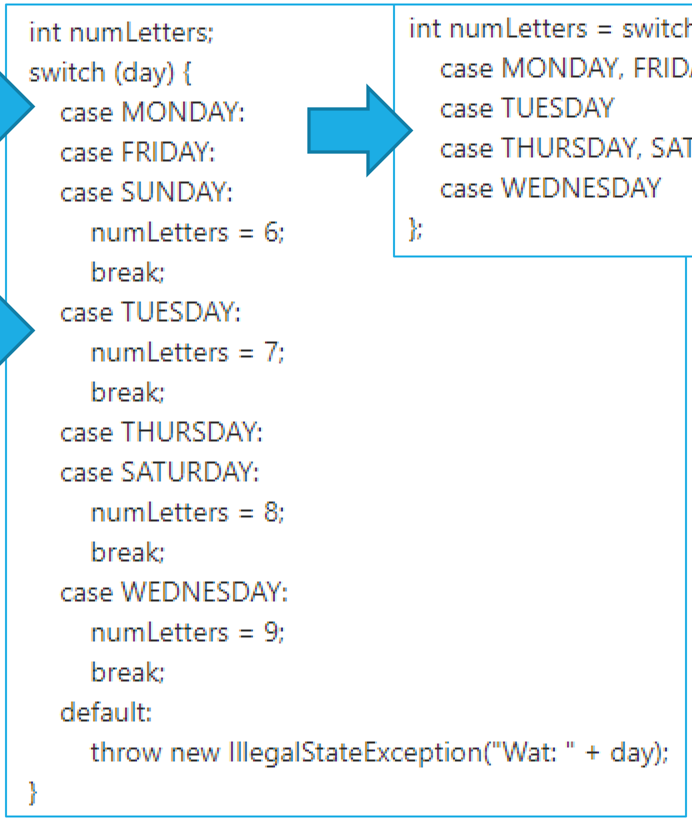
- Java 9
  - Modules (and “module-info.java”)
  - jShell (read-evaluate-print loop (REPL) tool)

- Java 10
  - Local-Variable Type Inference (skip type declaration associated with local variables).
  - Enhancements for garbage collection and compilation.

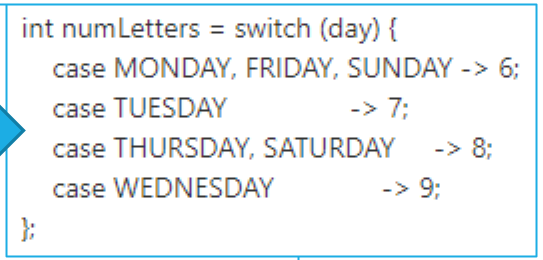
- Java 11
  - Removal of JavaFX, which is now available as a standalone technology.

What’s New in Java (highlights)

- Java 12
  - New garbage collector
  - New switch expression (preview)
- Java 13
  - Switch expressions (second preview)
  - Text blocks (preview)
- Java 14
  - Text Blocks (second preview)
  - Records (preview)



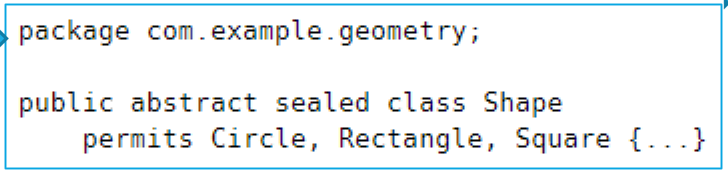
```
int numLetters;
switch (day) {
  case MONDAY:
  case FRIDAY:
  case SUNDAY:
    numLetters = 6;
    break;
  case TUESDAY:
    numLetters = 7;
    break;
  case THURSDAY:
  case SATURDAY:
    numLetters = 8;
    break;
  case WEDNESDAY:
    numLetters = 9;
    break;
  default:
    throw new IllegalStateException("Wat: " + day);
}
```



```
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> 6;
    case TUESDAY                -> 7;
    case THURSDAY, SATURDAY     -> 8;
    case WEDNESDAY              -> 9;
};
```

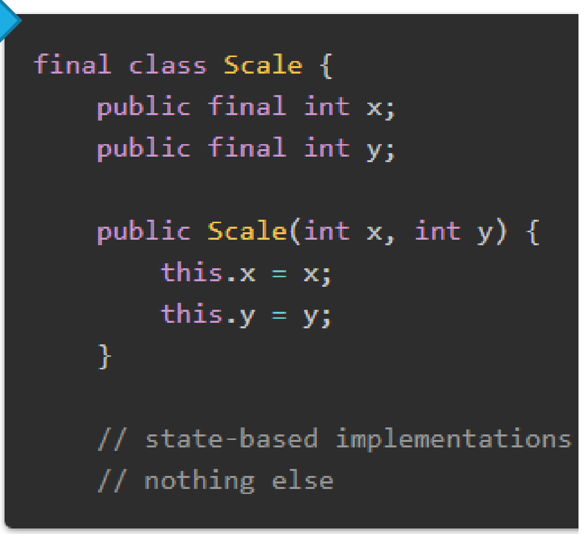
What’s New in Java (highlights)

- Java 15
  - Pattern matching for instanceof (second preview)
  - Records (second preview)
  - Sealed Classes (preview)
  - Text Blocks (standard)



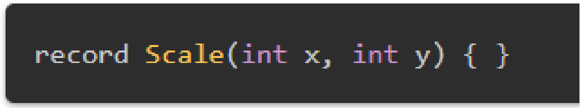
```
package com.example.geometry;

public abstract sealed class Shape
    permits Circle, Rectangle, Square {...}
```

Before:

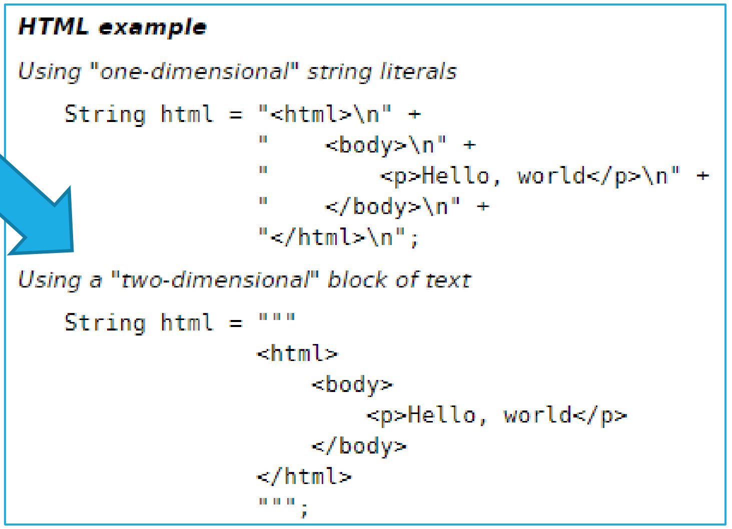


```
final class Scale {
    public final int x;
    public final int y;

    public Scale(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // state-based implementations
    // nothing else
```

Java 14 with records



```
record Scale(int x, int y) { }
```



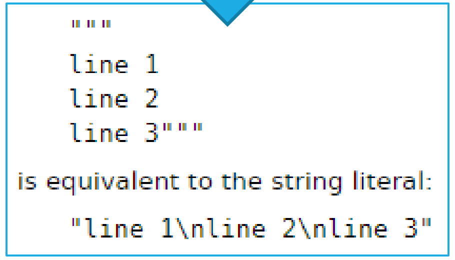
HTML example

Using "one-dimensional" string literals

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

Using a "two-dimensional" block of text

```
String html = """
              <html>
                  <body>
                      <p>Hello, world</p>
                  </body>
              </html>
              """;
```



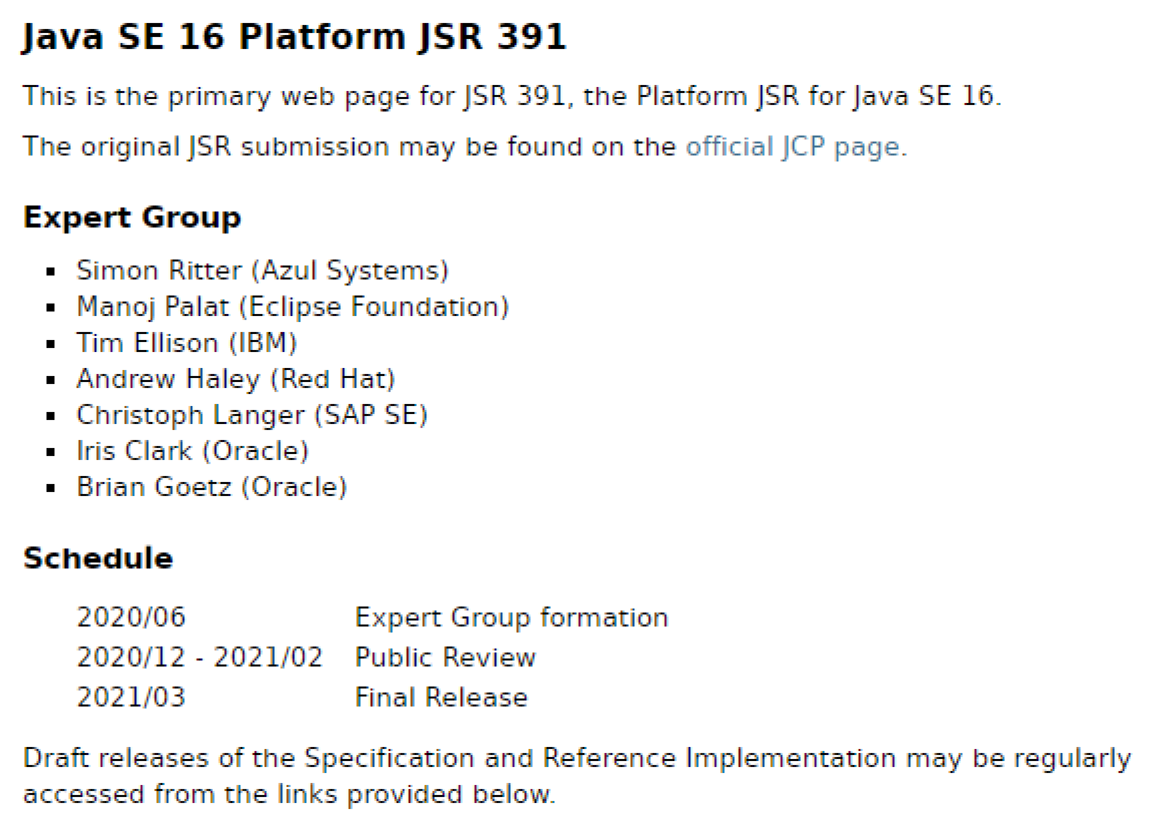
```
"""
line 1
line 2
line 3"""
```

is equivalent to the string literal:

```
"line 1\nline 2\nline 3"
```

What’s New in Java (highlights)

- Java 16



Java SE 16 Platform JSR 391

This is the primary web page for JSR 391, the Platform JSR for Java SE 16.

The original JSR submission may be found on the official JCP page.

Expert Group

- Simon Ritter (Azul Systems)
- Manoj Palat (Eclipse Foundation)
- Tim Ellison (IBM)
- Andrew Haley (Red Hat)
- Christoph Langer (SAP SE)
- Iris Clark (Oracle)
- Brian Goetz (Oracle)

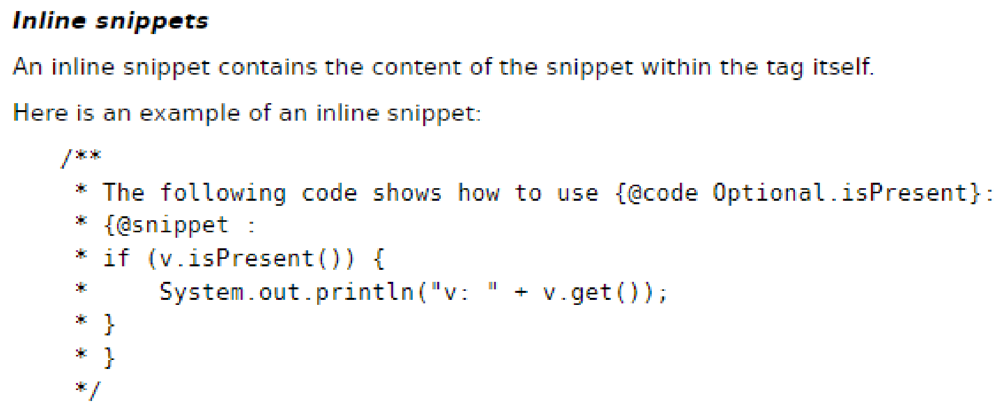
Schedule

2020/06	Expert Group formation
2020/12 - 2021/02	Public Review
2021/03	Final Release

Draft releases of the Specification and Reference Implementation may be regularly accessed from the links provided below.

What’s New in Java (highlights)

- Java 17
  - Has been superseded.
  - The current version do not include the most up to date security vulnerability fixes and no longer recommended for use in production.
- Java 18

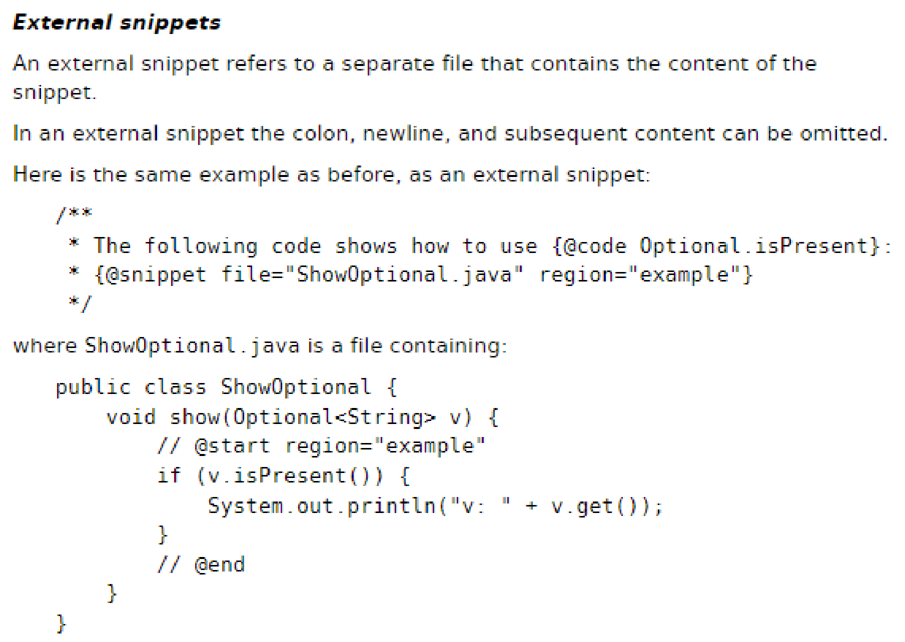


Inline snippets

An inline snippet contains the content of the snippet within the tag itself.

Here is an example of an inline snippet:

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet :
 * if (v.isPresent()) {
 *     System.out.println("v: " + v.get());
 * }
 * }
 */
```



External snippets

An external snippet refers to a separate file that contains the content of the snippet.

In an external snippet the colon, newline, and subsequent content can be omitted.

Here is the same example as before, as an external snippet:

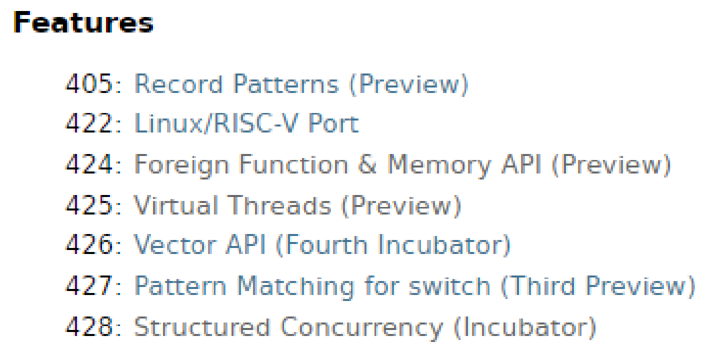
```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet file="ShowOptional.java" region="example"}
 */
```

where `ShowOptional.java` is a file containing:

```
public class ShowOptional {
    void show(Optional<String> v) {
        // @start region="example"
        if (v.isPresent()) {
            System.out.println("v: " + v.get());
        }
        // @end
    }
}
```

What’s New in Java (highlights)

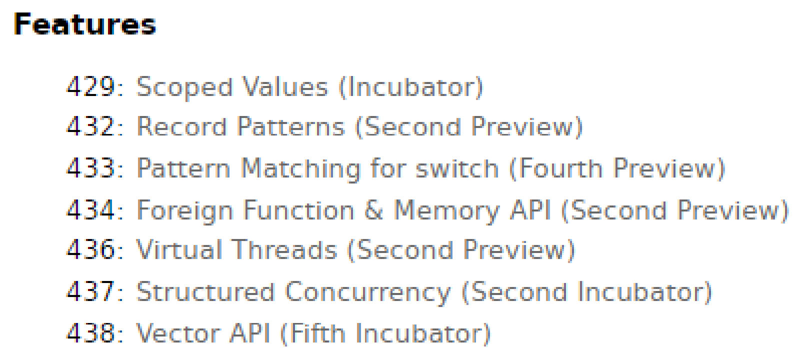
- Java 19
  - Released on September 20th, 2022.



Features

405: Record Patterns (Preview)
422: Linux/RISC-V Port
424: Foreign Function & Memory API (Preview)
425: Virtual Threads (Preview)
426: Vector API (Fourth Incubator)
427: Pattern Matching for switch (Third Preview)
428: Structured Concurrency (Incubator)

- Java 20
  - Released on March 21st, 2023.



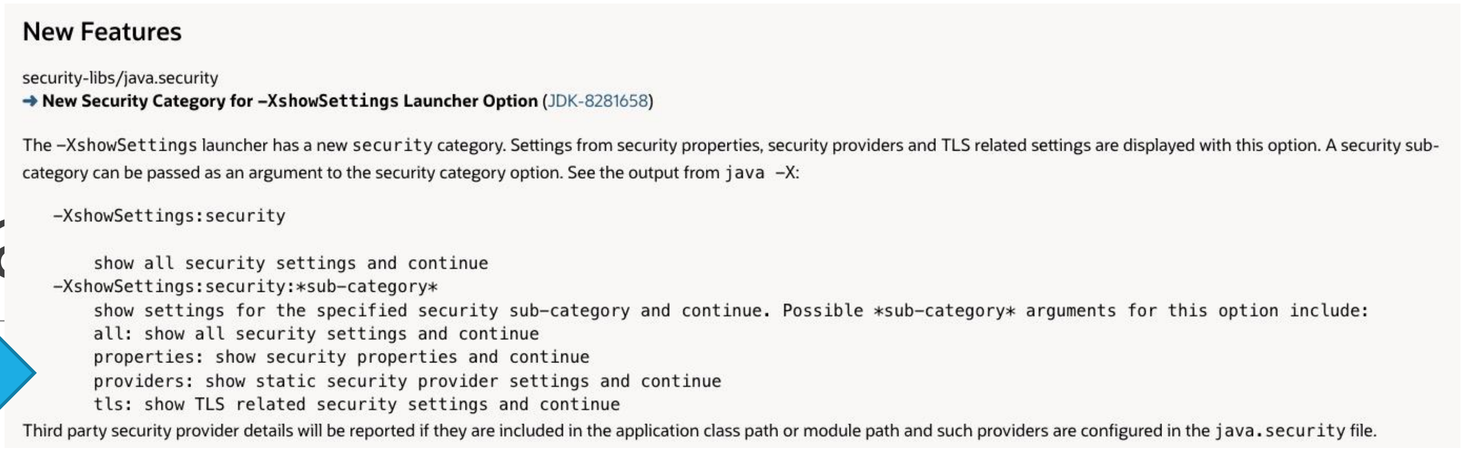
Features

429: Scoped Values (Incubator)
432: Record Patterns (Second Preview)
433: Pattern Matching for switch (Fourth Preview)
434: Foreign Function & Memory API (Second Preview)
436: Virtual Threads (Second Preview)
437: Structured Concurrency (Second Incubator)
438: Vector API (Fifth Incubator)

What’s New in Ja

- Java 21
  - Released on July 16th, 2024.

- Java 23
  - Released on September 17th, 2024.



New Features

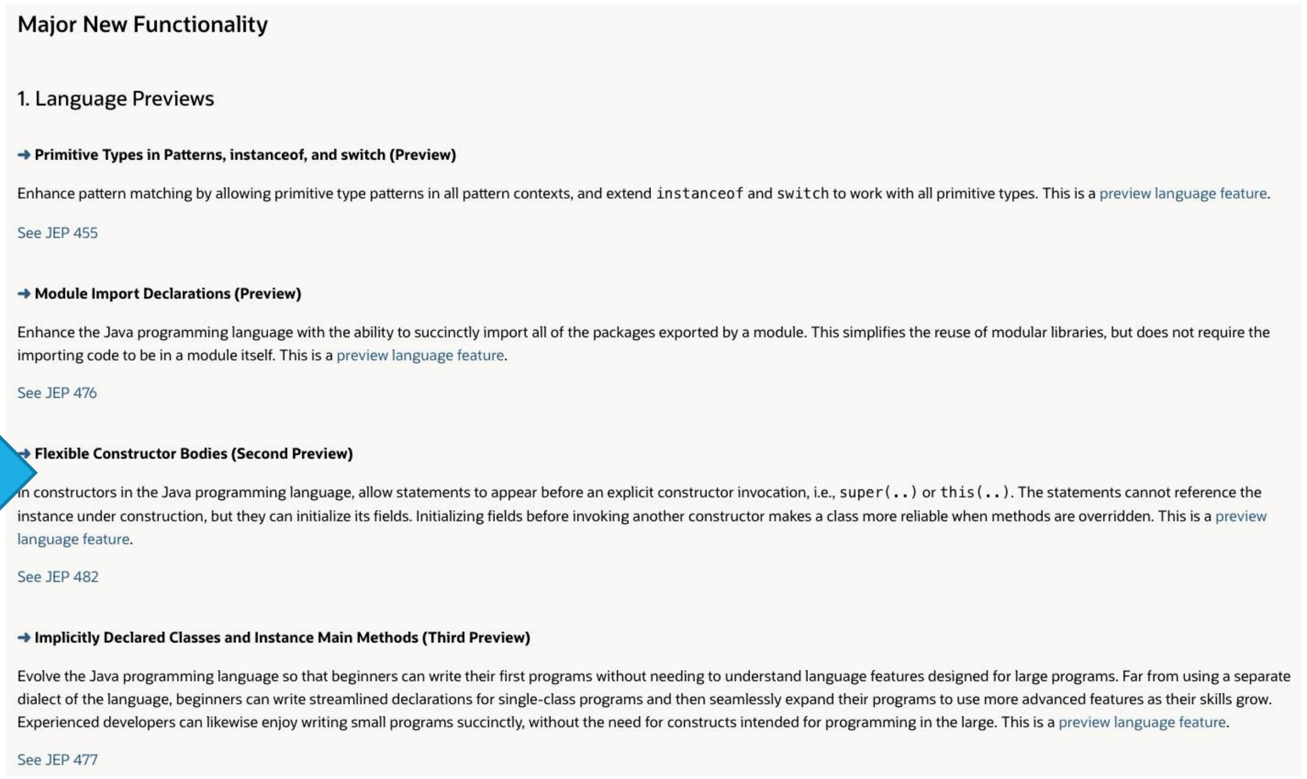
security-libs/java.security

➜ **New Security Category for `-XshowSettings` Launcher Option** (JDK-8281658)

The `-XshowSettings` launcher has a new `security` category. Settings from security properties, security providers and TLS related settings are displayed with this option. A security sub-category can be passed as an argument to the security category option. See the output from `java -X`:

```
-XshowSettings:security

    show all security settings and continue
-XshowSettings:security:*sub-category*
    show settings for the specified security sub-category and continue. Possible *sub-category* arguments for this option include:
    all: show all security settings and continue
    properties: show security properties and continue
    providers: show static security provider settings and continue
    tls: show TLS related security settings and continue
```

Third party security provider details will be reported if they are included in the application class path or module path and such providers are configured in the `java.security` file.



Major New Functionality

1. Language Previews

➜ **Primitive Types in Patterns, instanceof, and switch (Preview)**

Enhance pattern matching by allowing primitive type patterns in all pattern contexts, and extend `instanceof` and `switch` to work with all primitive types. This is a preview language feature.

See JEP 455

➜ **Module Import Declarations (Preview)**

Enhance the Java programming language with the ability to succinctly import all of the packages exported by a module. This simplifies the reuse of modular libraries, but does not require the importing code to be in a module itself. This is a preview language feature.

See JEP 476

➜ **Flexible Constructor Bodies (Second Preview)**

In constructors in the Java programming language, allow statements to appear before an explicit constructor invocation, i.e., `super(..)` or `this(..)`. The statements cannot reference the instance under construction, but they can initialize its fields. Initializing fields before invoking another constructor makes a class more reliable when methods are overridden. This is a preview language feature.

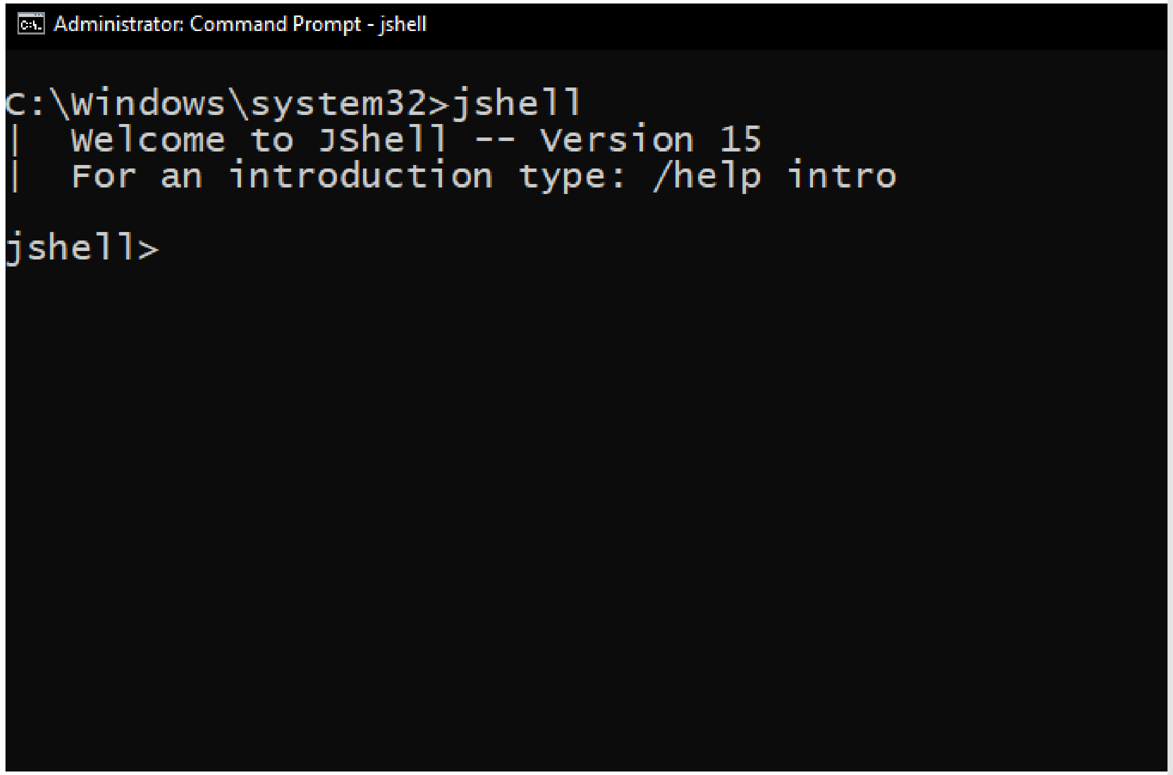
See JEP 482

➜ **Implicitly Declared Classes and Instance Main Methods (Third Preview)**

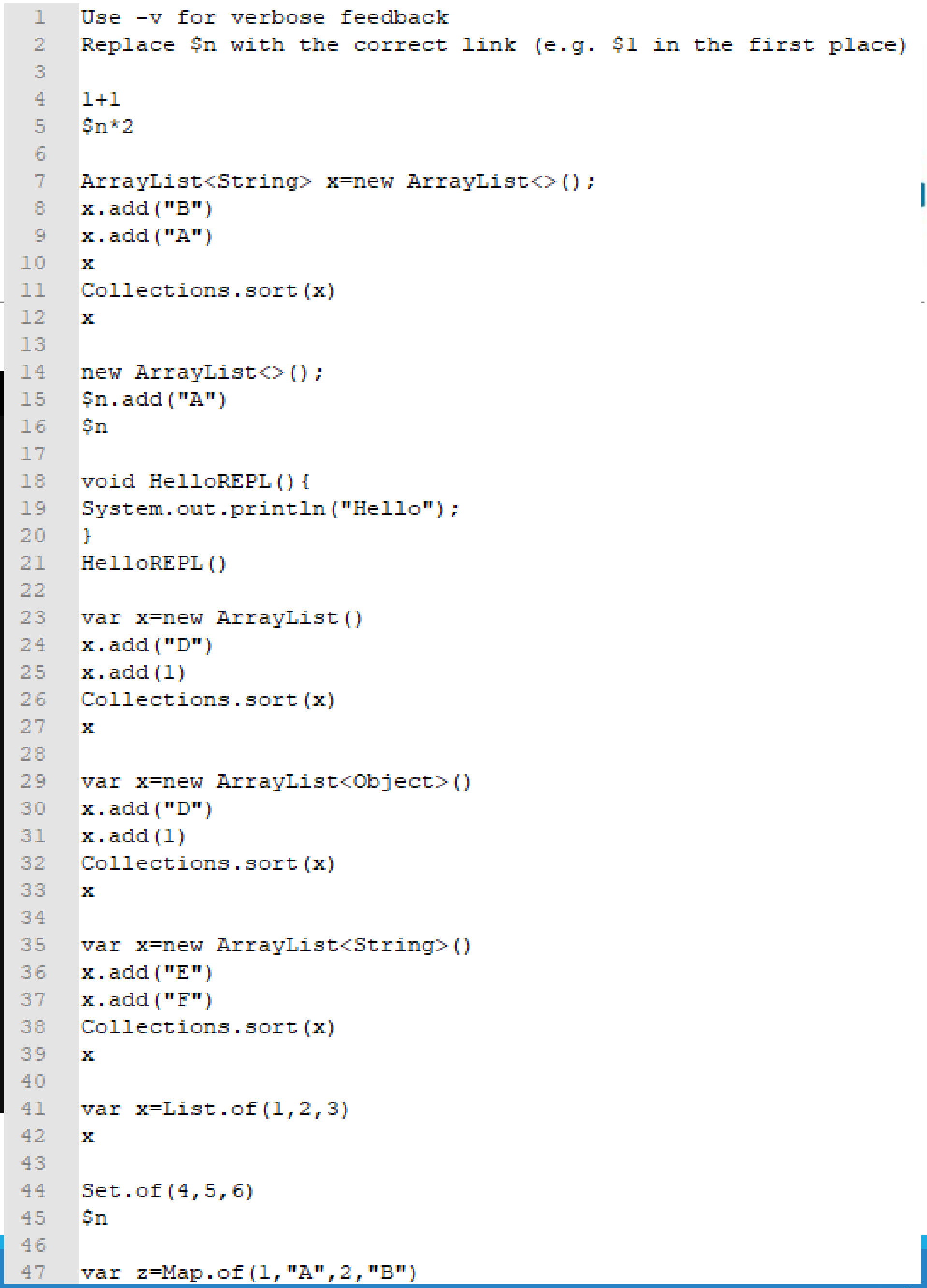
Evolve the Java programming language so that beginners can write their first programs without needing to understand language features designed for large programs. Far from using a separate dialect of the language, beginners can write streamlined declarations for single-class programs and then seamlessly expand their programs to use more advanced features as their skills grow. Experienced developers can likewise enjoy writing small programs succinctly, without the need for constructs intended for programming in the large. This is a preview language feature.

See JEP 477

jShell – Demonstration

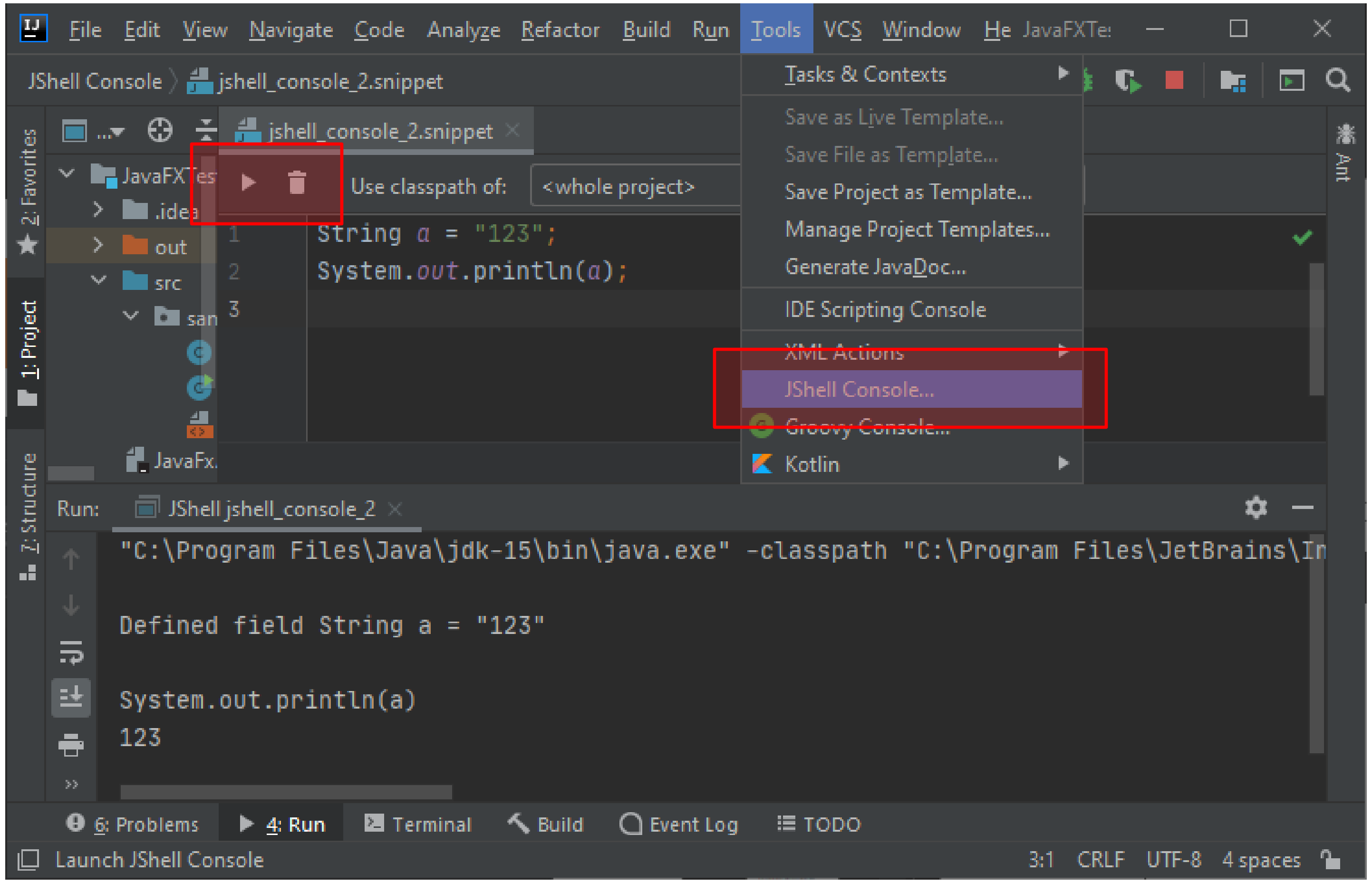


```
C:\Windows\system32>jshell
|  Welcome to JShell -- Version 15
|  For an introduction type: /help intro

jshell>
```



```
Use -v for verbose feedback
Replace $n with the correct link (e.g. $1 in the first place)

1+1
$n*2

ArrayList<String> x=new ArrayList<>();
x.add("B")
x.add("A")
x
Collections.sort(x)
x

new ArrayList<>();
$n.add("A")
$n

void HelloREPL(){
System.out.println("Hello");
}
HelloREPL()

var x=new ArrayList()
x.add("D")
x.add(1)
Collections.sort(x)
x

var x=new ArrayList<Object>()
x.add("D")
x.add(1)
Collections.sort(x)
x

var x=new ArrayList<String>()
x.add("E")
x.add("F")
Collections.sort(x)
x

var x=List.of(1,2,3)
x

Set.of(4,5,6)
$n

var z=Map.of(1,"A",2,"B")
```

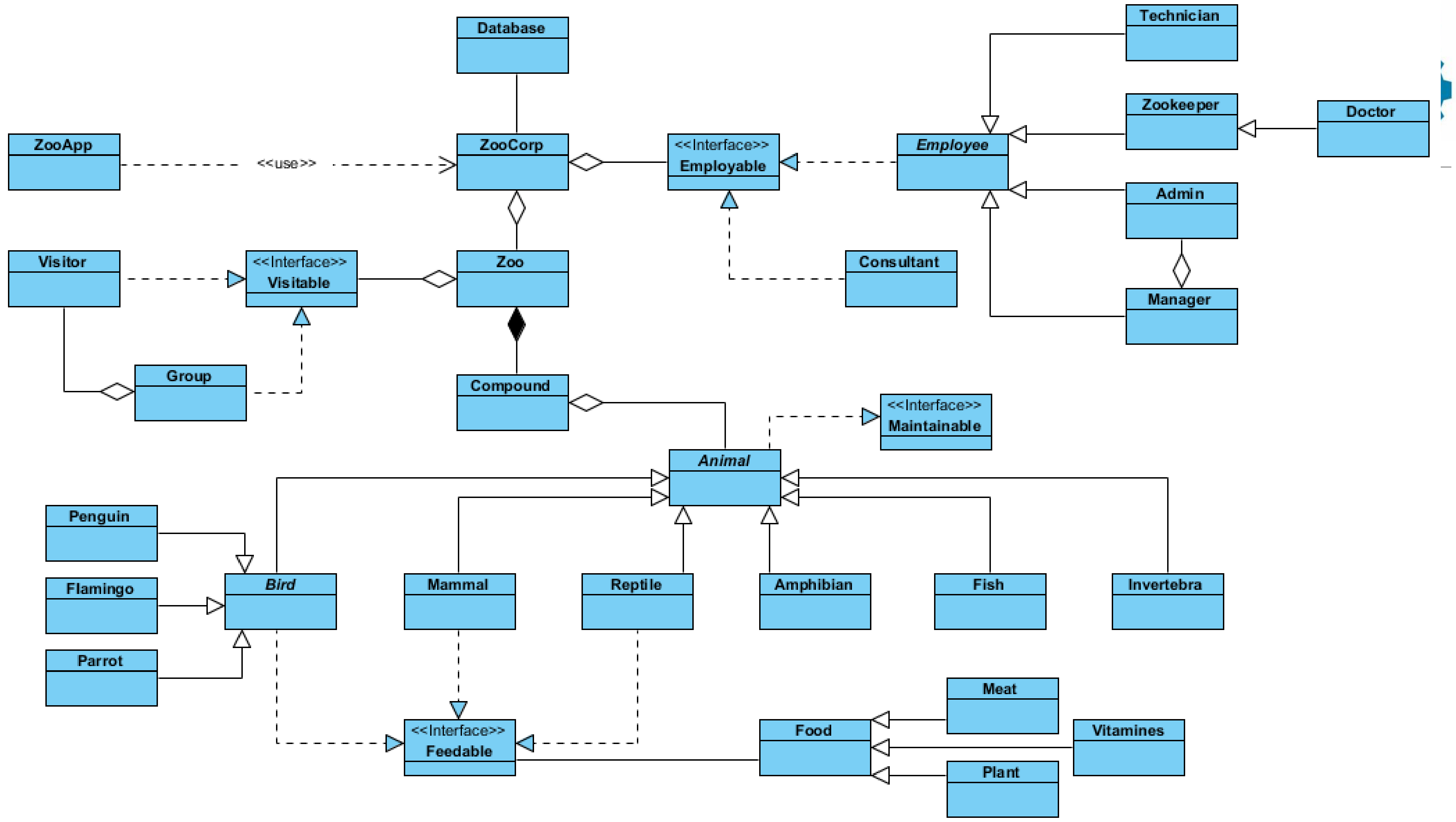
jShell – IntelliJ Demonstration


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window He JavaFXTe
JShell Console jshell_console_2.snippet
Tasks & Contexts
Save as Live Template...
Save File as Template...
Save Project as Template...
Manage Project Templates...
Generate JavaDoc...
IDE Scripting Console
XML Actions
JShell Console...
Groovy Console...
Kotlin
Use classpath of: <whole project>
Run: JShell jshell_console_2
6: Problems 4: Run Terminal Build Event Log TODO
Launch JShell Console
3:1 CRLF UTF-8 4 spaces

```
String a = "123";
System.out.println(a);
```

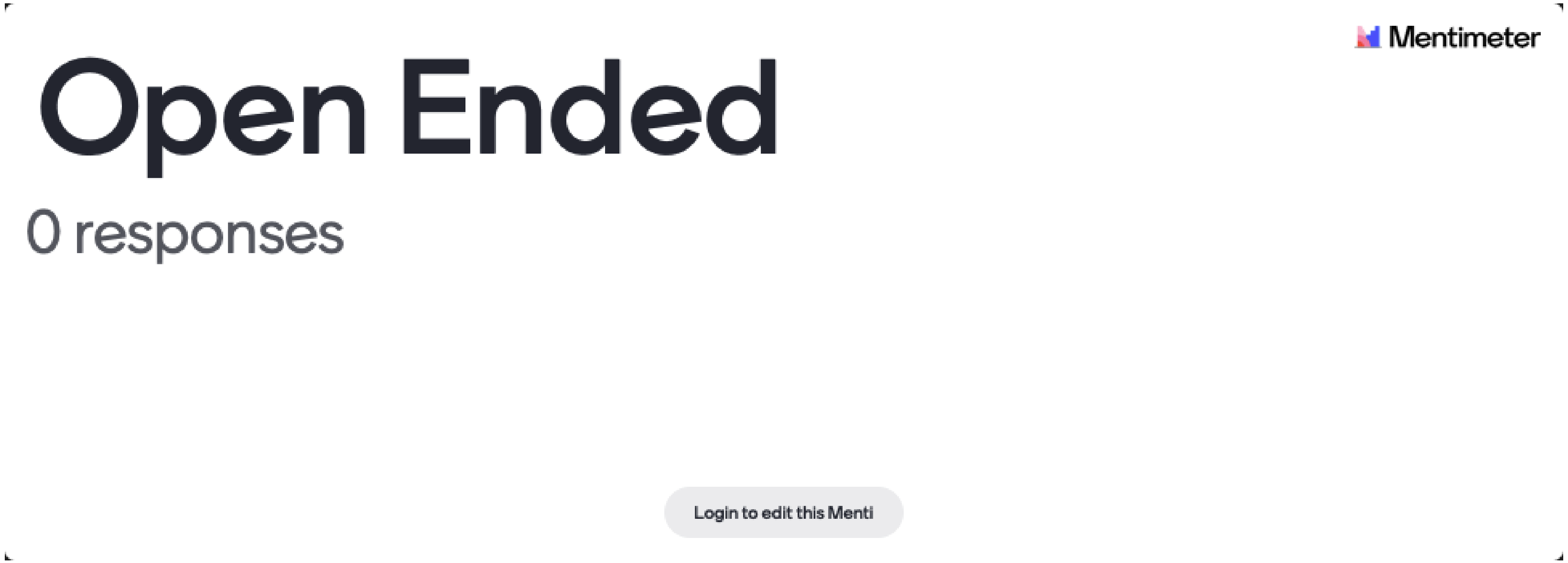
```
"C:\Program Files\Java\jdk-15\bin\java.exe" -classpath "C:\Program Files\JetBrains\In

Defined field String a = "123"

System.out.println(a)
123
```

Maintaining the Zoo Management Software


Database
ZooApp
<<use>>
ZooCorp
<<Interface>> Employable
Employee
Technician
Zookeeper
Doctor
Admin
Manager
Consultant
Visitor
<<Interface>> Visitable
Zoo
Group
Compound
<<Interface>> Maintainable
Animal
Penguin
Flamingo
Parrot
Bird
Mammal
Reptile
Amphibian
Fish
Invertebra
<<Interface>> Feedable
Food
Meat
Vitamines
Plant

Basic Maintenance

- What would you propose how we should get started?



Mentimeter

Open Ended

0 responses

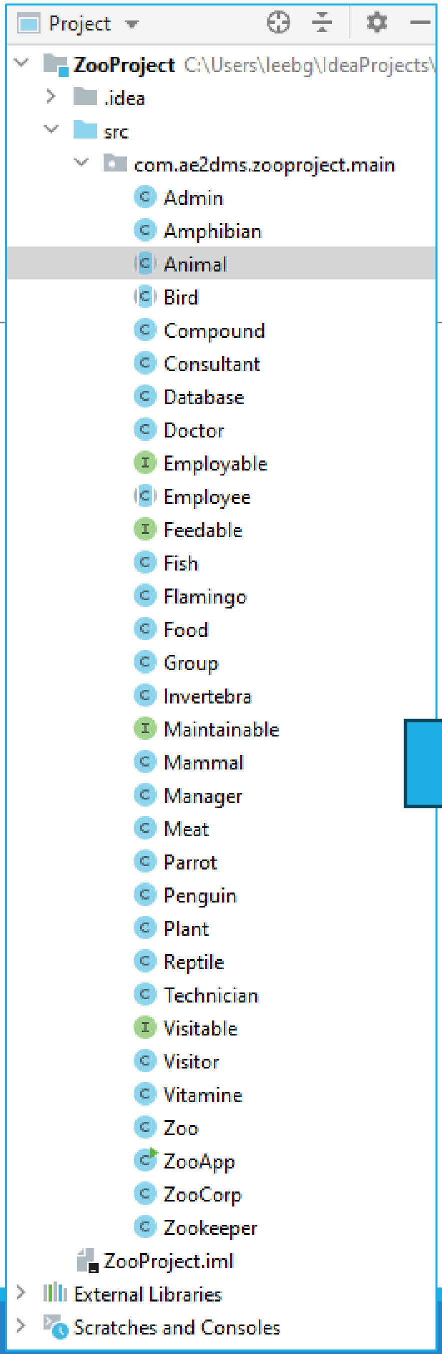
Basic Maintenance

- Dividing all classes into packages.
  - Why is this useful?
    - Providing individual name spaces.
    - Making large project easier to handle (better organised).
    - Dividing responsibilities amongst colleagues.
- Making all animal subclass abstract.
  - They should not be instantiated as they are still high level.
- Removing redundant code.
  - “buildNewCompounds()” is not required anymore.
  - Remove some setters (to gain control).
- Commenting code and producing Javadocs.

Basic Maintenance

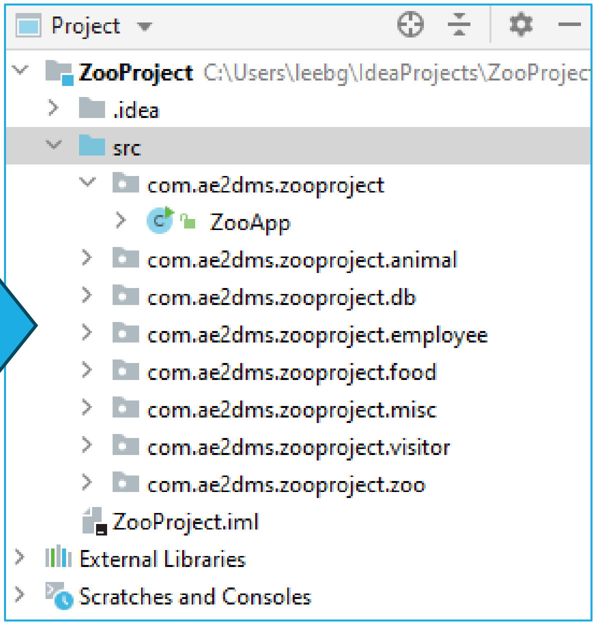
- Packages are used in Java in order to prevent naming conflicts, to control access, to make searching, locating and usage of classes, interfaces, enumerations and annotations easier etc.

Unstructured



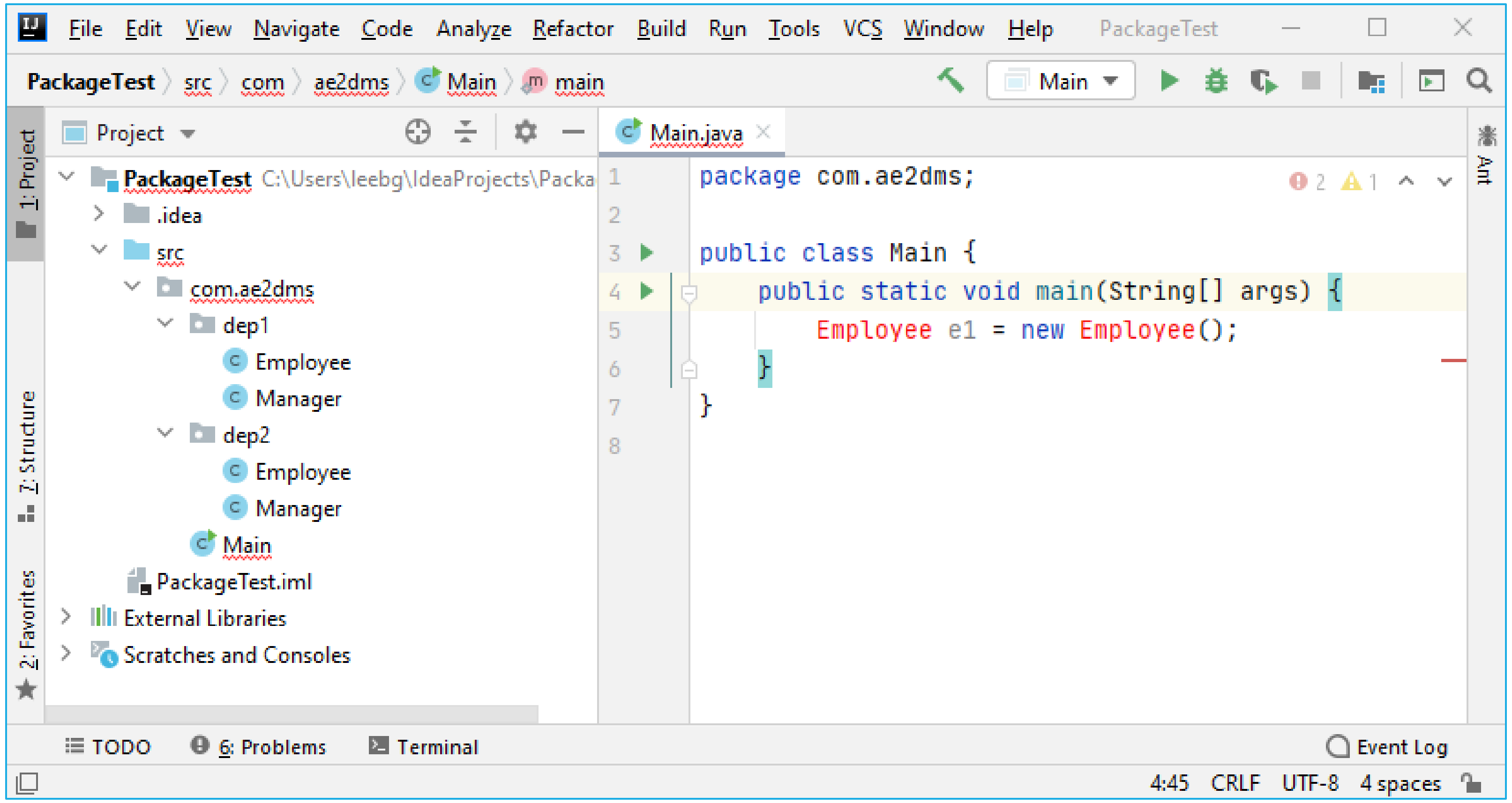
```
Project
ZooProject C:\Users\leebg\IdeaProjects\
  .idea
  src
    com.ae2dms.zooproject.main
      Admin
      Amphibian
      Animal
      Bird
      Compound
      Consultant
      Database
      Doctor
      Employable
      Employee
      Feedable
      Fish
      Flamingo
      Food
      Group
      Invertebra
      Maintainable
      Mammal
      Manager
      Meat
      Parrot
      Penguin
      Plant
      Reptile
      Technician
      Visitable
      Visitor
      Vitamine
      Zoo
      ZooApp
      ZooCorp
      Zookeeper
    ZooProject.iml
External Libraries
Scratches and Consoles
```

Restructured



```
Project
ZooProject C:\Users\leebg\IdeaProjects\ZooProjec
  .idea
  src
    com.ae2dms.zooproject
      ZooApp
    com.ae2dms.zooproject.animal
    com.ae2dms.zooproject.db
    com.ae2dms.zooproject.employee
    com.ae2dms.zooproject.food
    com.ae2dms.zooproject.misc
    com.ae2dms.zooproject.visitor
    com.ae2dms.zooproject.zoo
  ZooProject.iml
External Libraries
Scratches and Consoles
```

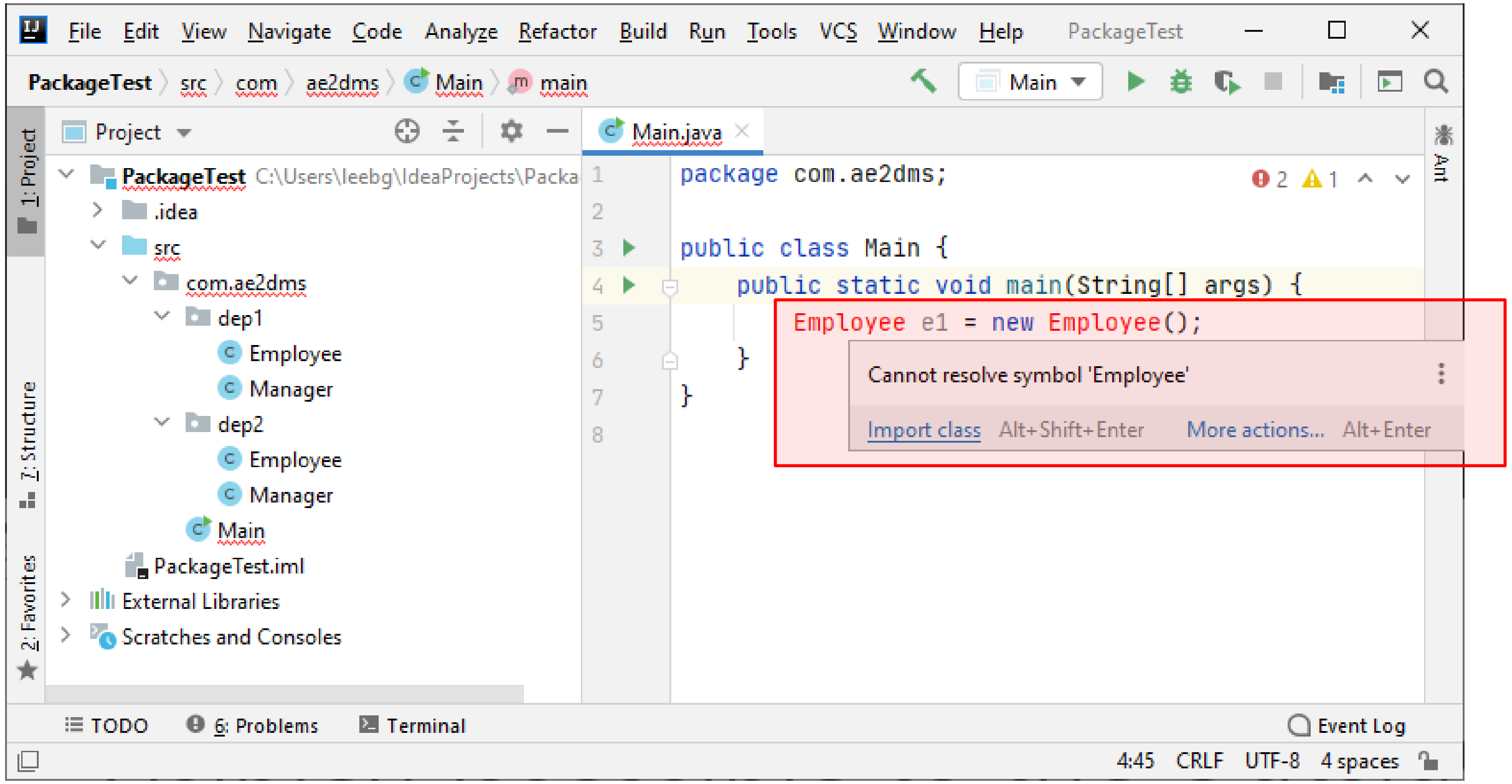
Communication Between Packages



```
PackageTest C:\Users\leebg\IdeaProjects\Packa
  .idea
  src
    com.ae2dms
      dep1
        Employee
        Manager
      dep2
        Employee
        Manager
      Main
    PackageTest.iml
External Libraries
Scratches and Consoles
```

```java
package com.ae2dms;

public class Main {
    public static void main(String[] args) {
        Employee e1 = new Employee();
    }
}
```

Modularisation: Deciding What is Visible/Accessible to the Outside World

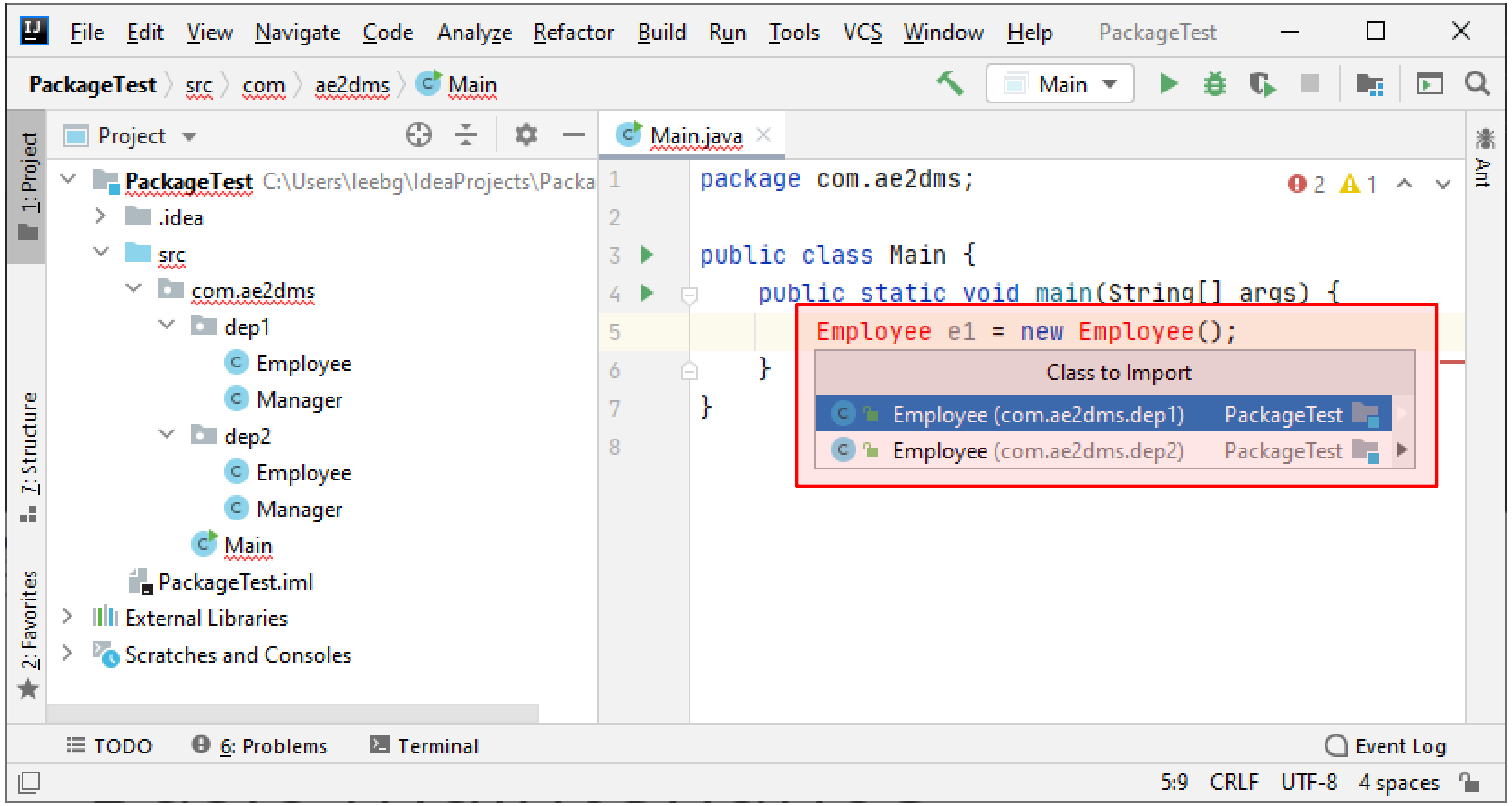


```
package com.ae2dms;

public class Main {
    public static void main(String[] args) {
        Employee e1 = new Employee();
    }
}
```

Cannot resolve symbol 'Employee'

Import class Alt+Shift+Enter More actions... Alt+Enter

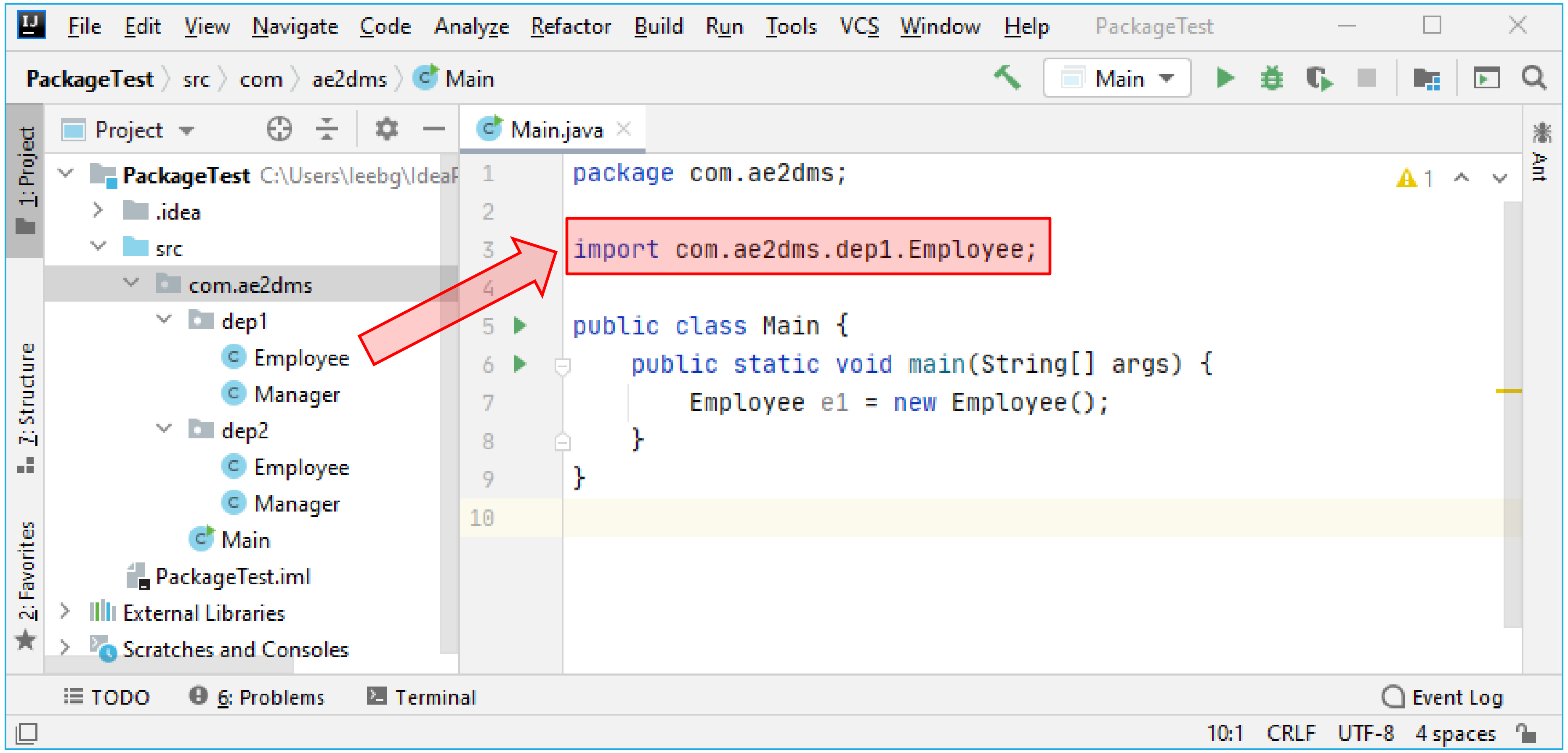
Modularisation: Deciding What is Visible/Accessible to the Outside World


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help PackageTest
PackageTest src com ae2dms Main
Project
PackageTest C:\Users\leebg\IdeaProjects\Packa
.idea
src
com.ae2dms
dep1
Employee
Manager
dep2
Employee
Manager
Main
PackageTest.iml
External Libraries
Scratches and Consoles
Main.java
TODO 6: Problems Terminal Event Log
5:9 CRLF UTF-8 4 spaces

```
package com.ae2dms;

public class Main {
    public static void main(String[] args) {
        Employee e1 = new Employee();
    }
}
```

Class to Import
Employee (com.ae2dms.dep1) PackageTest
Employee (com.ae2dms.dep2) PackageTest

Modularisation: Deciding What is Visible/Accessible to the Outside World



```java
package com.ae2dms;

import com.ae2dms.dep1.Employee;

public class Main {
    public static void main(String[] args) {
        Employee e1 = new Employee();
    }
}
```

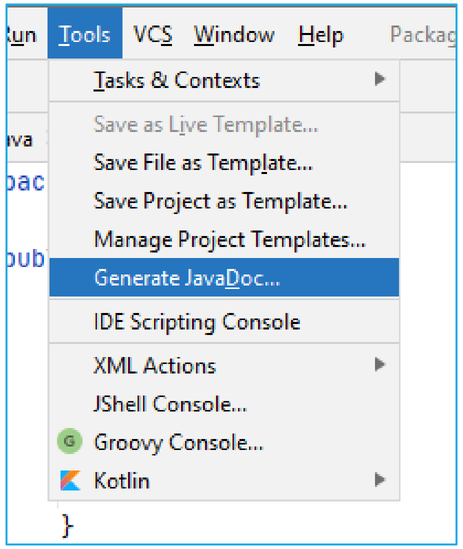
Basic Maintenance

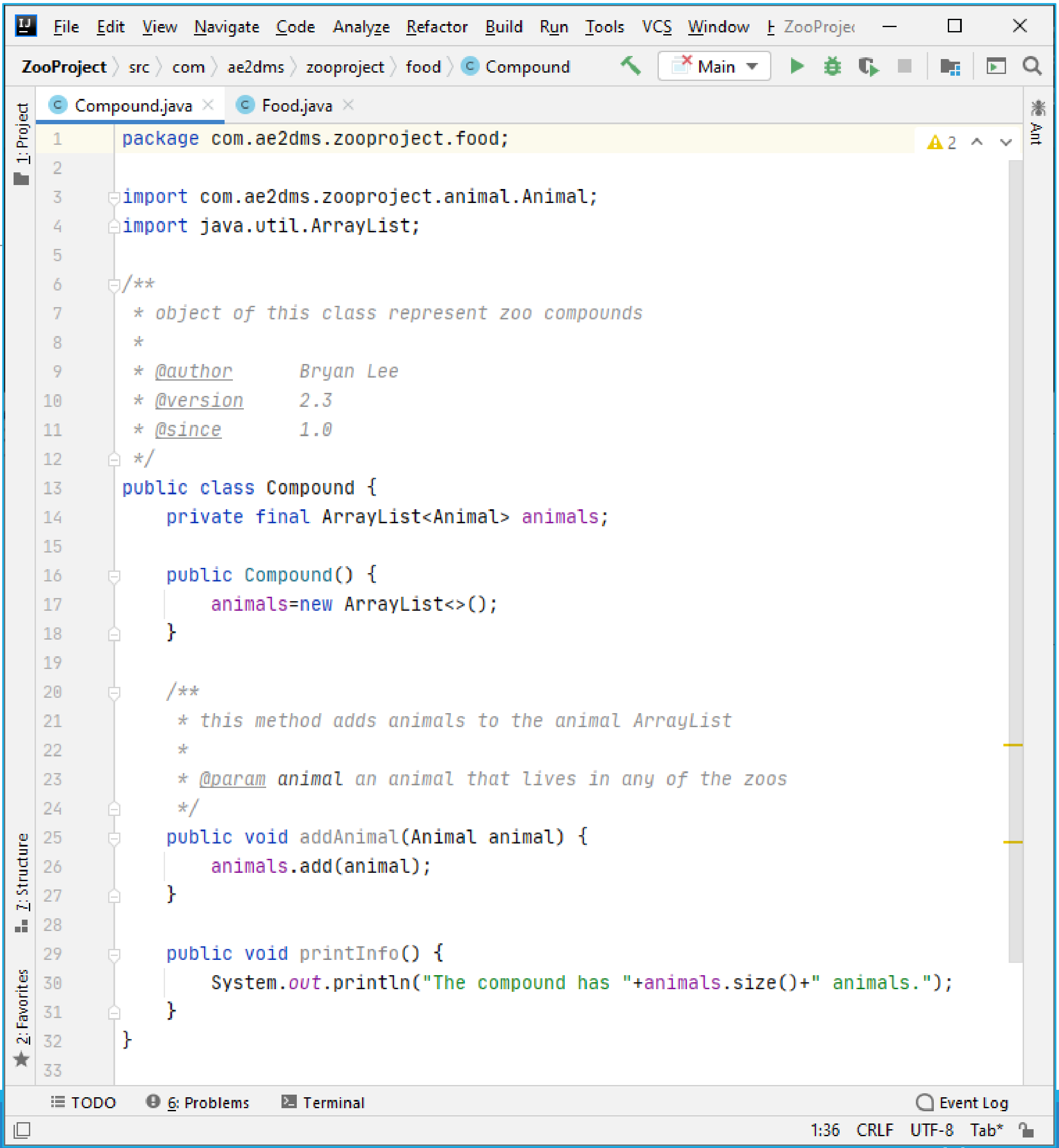
- Javadocs

```
/**
* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
* <p>
* xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
* @paramvariable Description.
* @return Description.
*/
```

@author John Smith Class, Interface, Enum
@version versionClass, Interface, Enum
@since since-text Class, Interface, Enum, Field, Method
@see reference Class, Interface, Enum, Field, Method
@paramname description Method
@return description Method
@deprecated description Class, Interface, Enum, Field, Method

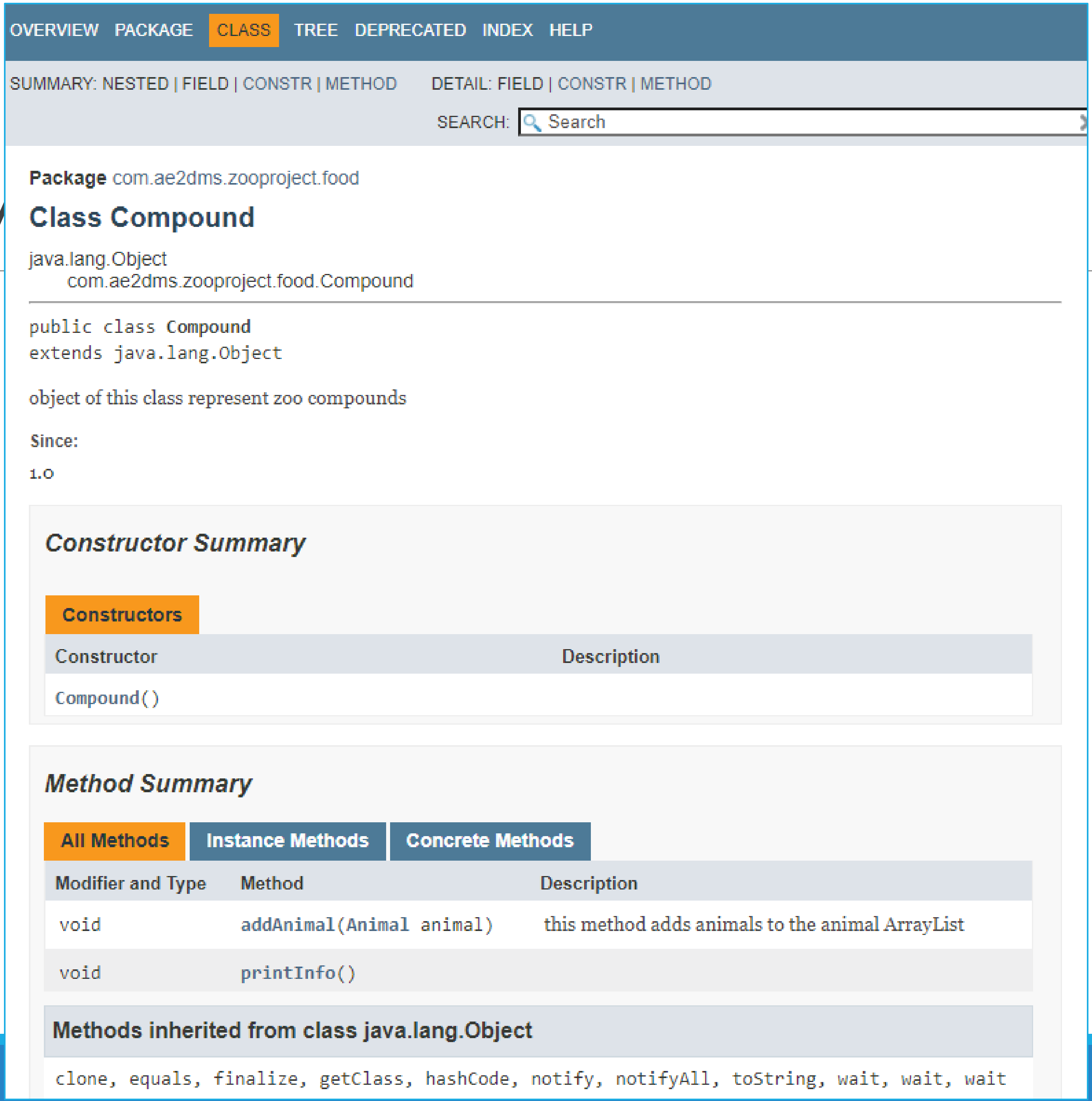
Basic Maintenance




File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window ZooProject
ZooProject src com ae2dms zooproject food Compound
Main
Compound.java
Food.java
TODO
6: Problems
Terminal
Event Log
1:36 CRLF UTF-8 Tab*

```java
package com.ae2dms.zooproject.food;

import com.ae2dms.zooproject.animal.Animal;
import java.util.ArrayList;

/**
 * object of this class represent zoo compounds
 *
 * @author      Bryan Lee
 * @version     2.3
 * @since       1.0
 */
public class Compound {
    private final ArrayList<Animal> animals;

    public Compound() {
        animals=new ArrayList<>();
    }

    /**
     * this method adds animals to the animal ArrayList
     *
     * @param animal an animal that lives in any of the zoos
     */
    public void addAnimal(Animal animal) {
        animals.add(animal);
    }

    public void printInfo() {
        System.out.println("The compound has "+animals.size()+" animals.");
    }
}
```

Basic M



OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

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

SEARCH: Search

Package com.ae2dms.zooproject.food

Class Compound

java.lang.Object
    com.ae2dms.zooproject.food.Compound

```
public class Compound
extends java.lang.Object
```

object of this class represent zoo compounds

Since:

1.0

Constructor Summary

Constructors

Constructor	Description
`Compound()`	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`void`	`addAnimal(Animal animal)`	this method adds animals to the animal ArrayList
`void`	`printInfo()`	

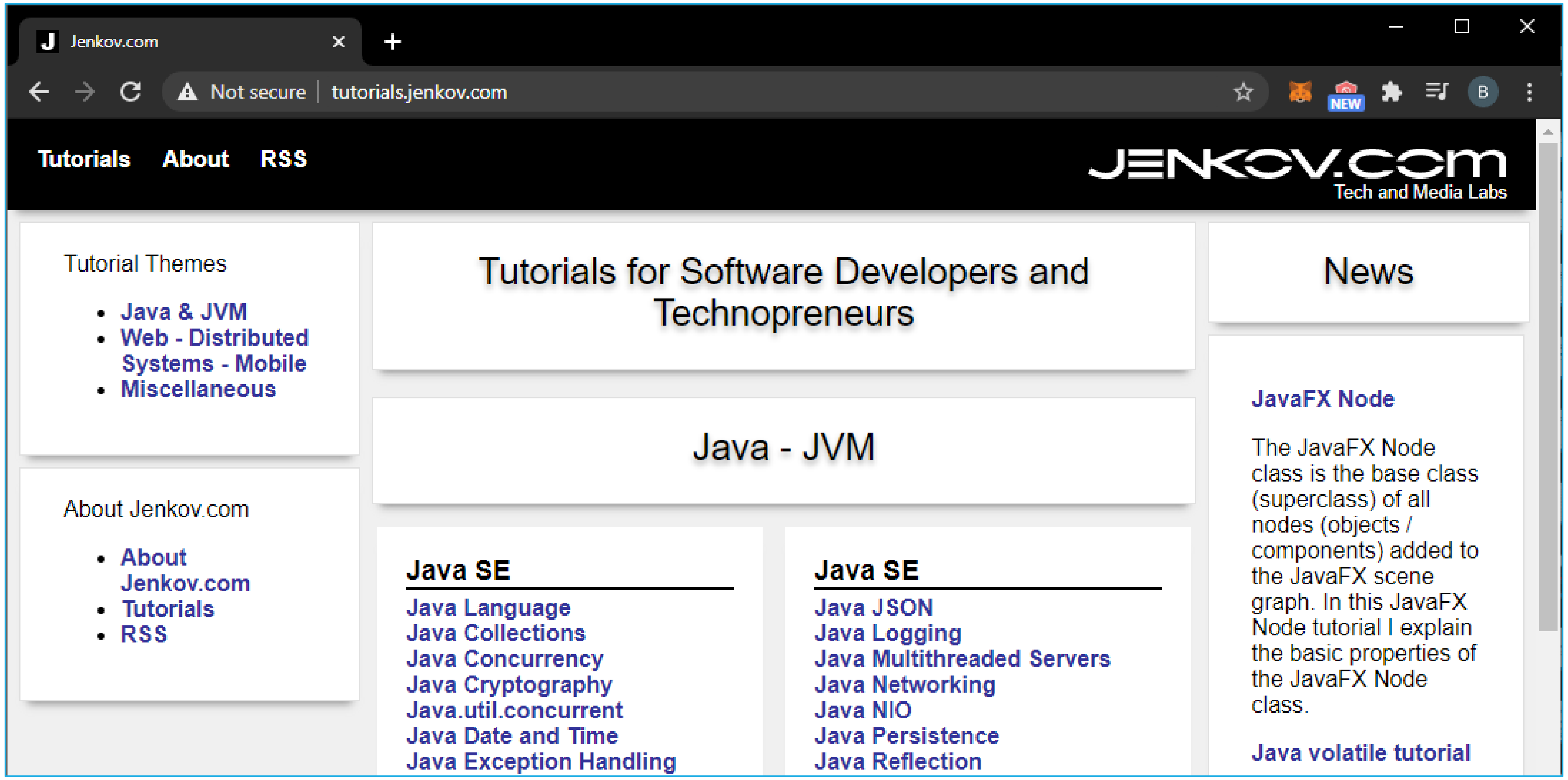
Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Useful Resource

FOR STUDYING JAVA IN MORE DEPTH

Useful Resource: Jenkov.com



Jenkov.com

Not secure | tutorials.jenkov.com

Tutorials About RSS

JENKOV.COM Tech and Media Labs

Tutorial Themes

- Java & JVM
- Web - Distributed Systems - Mobile
- Miscellaneous

About Jenkov.com

- About Jenkov.com
- Tutorials
- RSS

Tutorials for Software Developers and Technopreneurs

Java - JVM

Java SE

Java Language
Java Collections
Java Concurrency
Java Cryptography
Java.util.concurrent
Java Date and Time
Java Exception Handling

Java SE

Java JSON
Java Logging
Java Multithreaded Servers
Java Networking
Java NIO
Java Persistence
Java Reflection

News

JavaFX Node

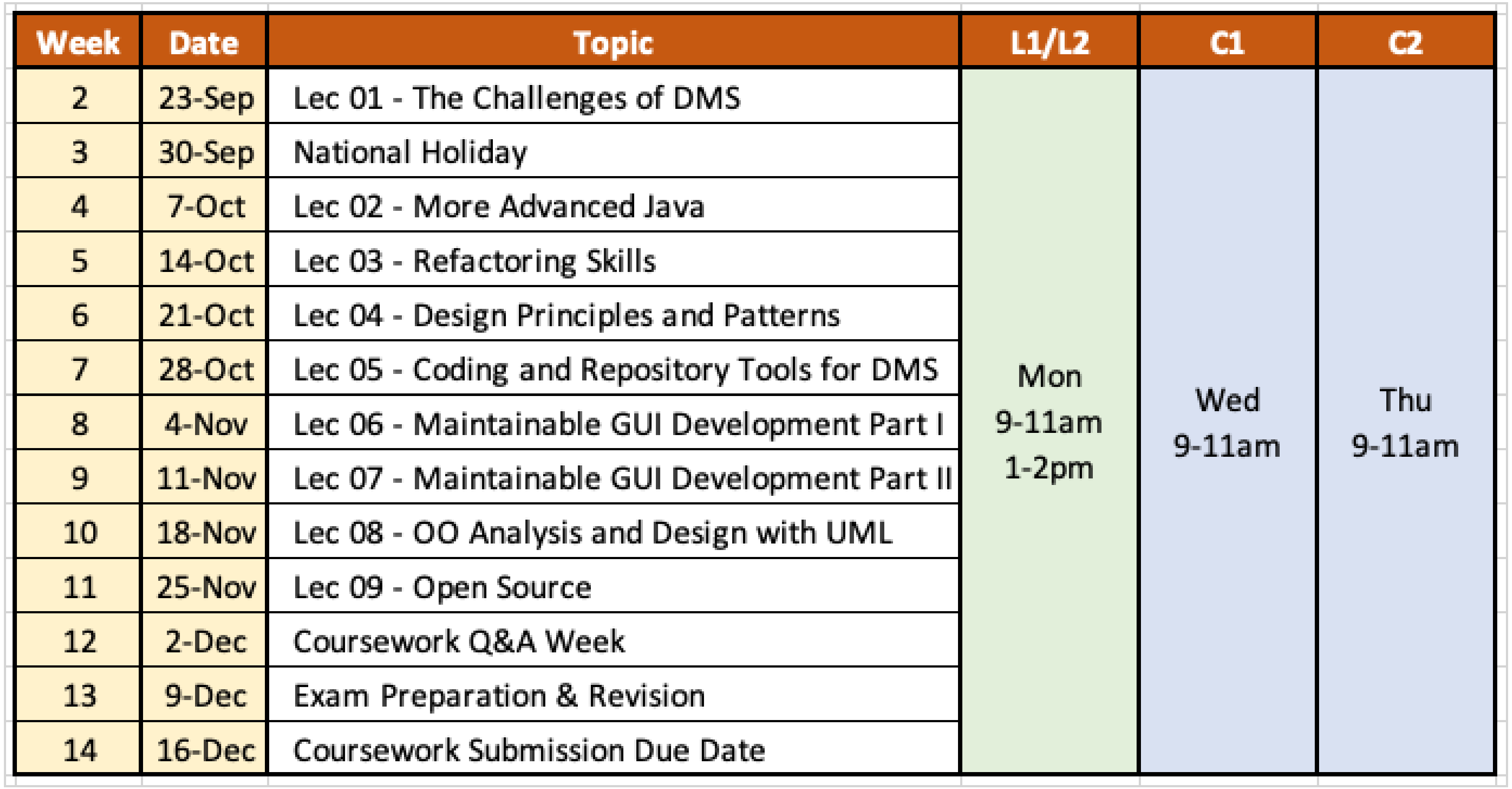
The JavaFX Node class is the base class (superclass) of all nodes (objects / components) added to the JavaFX scene graph. In this JavaFX Node tutorial I explain the basic properties of the JavaFX Node class.

Java volatile tutorial

Module Organisation

HTTPS://MOODLE.NOTTINGHAM.AC.UK/COURSE/VIEW.PHP?ID=117564

Timetable



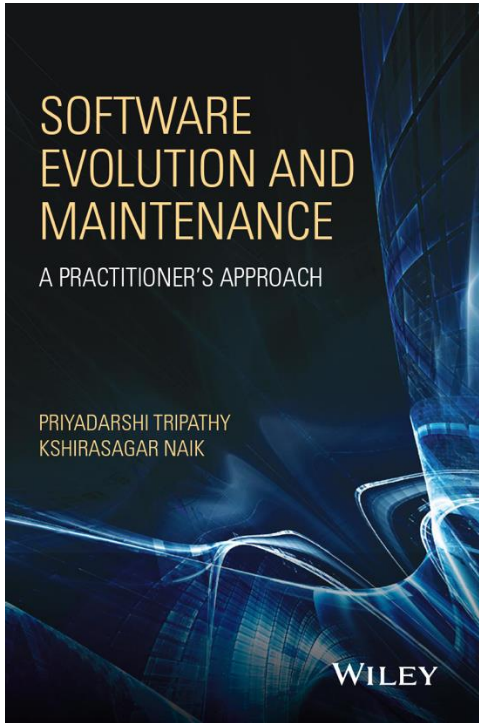
Week	Date	Topic	L1/L2	C1	C2
2	23-Sep	Lec 01 - The Challenges of DMS	Mon 9-11am 1-2pm	Wed 9-11am	Thu 9-11am
3	30-Sep	National Holiday			
4	7-Oct	Lec 02 - More Advanced Java			
5	14-Oct	Lec 03 - Refactoring Skills			
6	21-Oct	Lec 04 - Design Principles and Patterns			
7	28-Oct	Lec 05 - Coding and Repository Tools for DMS			
8	4-Nov	Lec 06 - Maintainable GUI Development Part I			
9	11-Nov	Lec 07 - Maintainable GUI Development Part II			
10	18-Nov	Lec 08 - OO Analysis and Design with UML			
11	25-Nov	Lec 09 - Open Source			
12	2-Dec	Coursework Q&A Week			
13	9-Dec	Exam Preparation & Revision			
14	16-Dec	Coursework Submission Due Date			

Module Assessments

- Exam – 25%
- Coursework – 75%
  - Modifying and extending existing code
  - See issue sheet for details.

Reference Textbook

- Software Evolution and Maintenance: A Practitioner’s Approach
  - Author: Tripathy, Priyadarshi; Naik, Kshirasagar
  - ISBN: 9780470603413
  - Publisher: John Wiley & Sons



Java SDK + IDE + JavaFX + Scene Builder

- JDK 16 / JDK 21 (for Windows, only x64 bit OS)
- JavaFX 16 SDK / JavaFX 21 SDK
  - Placed the “javafx-sdk-16” / “javafx-sdk-21” folder under “C:\Program Files\Java”.
  - MacOS – follow instructions HERE.
- Scene Builder 16.X.X / Scene Builder 23.X.X
  - The default location should be “C:\Program Files\SceneBuilder” for Windows.
- IntelliJ IDEA (**Community Edition (GRAY)** – FREE!)
  - Tick “Add launchers dir to the PATH”
  - Create a new JavaFX project – follow ALL the instructions in https://www.jetbrains.com/help/idea/javafx.html
  - Configure JavaFX Scene Builder – follow ALL the instructions in https://www.jetbrains.com/help/idea/opening-fxml-files-in-javafx-scene-builder.html
- **Troubleshooting:**
  - **Invalid path (for MacOS user): replace the %23 in the link with #**

Software

- Visual Paradigm Community Edition
- UNNC Gitlab
  - Link: https://csprojects.nottingham.edu.cn/

University of Nottingham
UK | CHINA | MALAYSIA

Module Modalities Work

- 20 credits = 200 hours of work
  - Lecture/Teaching: 30 hours
  - Computing Labs: 20 hours
  - Self Study: 30 hours
  - Coursework: 100 hours
  - Revision/Exam: 20 hours

University of Nottingham
UK | CHINA | MALAYSIA

Matters

- **Not a big fan for Q&A via emails** for several reasons:
  - Repeated/duplicate (same) questions from different students.
  - Difficulty to go through a list of unread inbox in UNNC email.
  - Took long time to read and reply.

- Merits of having **Q&A** in **Discussion Forum**:
  - Fast search to the similar issues without waiting for our reply.
  - Demonstrate your understanding of the lecture/practical lesson to your peers in the class.
  - Improve your technique in scientific expressions (important for GRP and FYP reports, and paper writing).
  - Create a learning environment for module discussion (**NOT** copy-paste/request/get solution for the coursework).
  - **IMPORTANT**: Please go through the **FAQ** before posting any questions.

Put Your Mind in Maintenance Mode