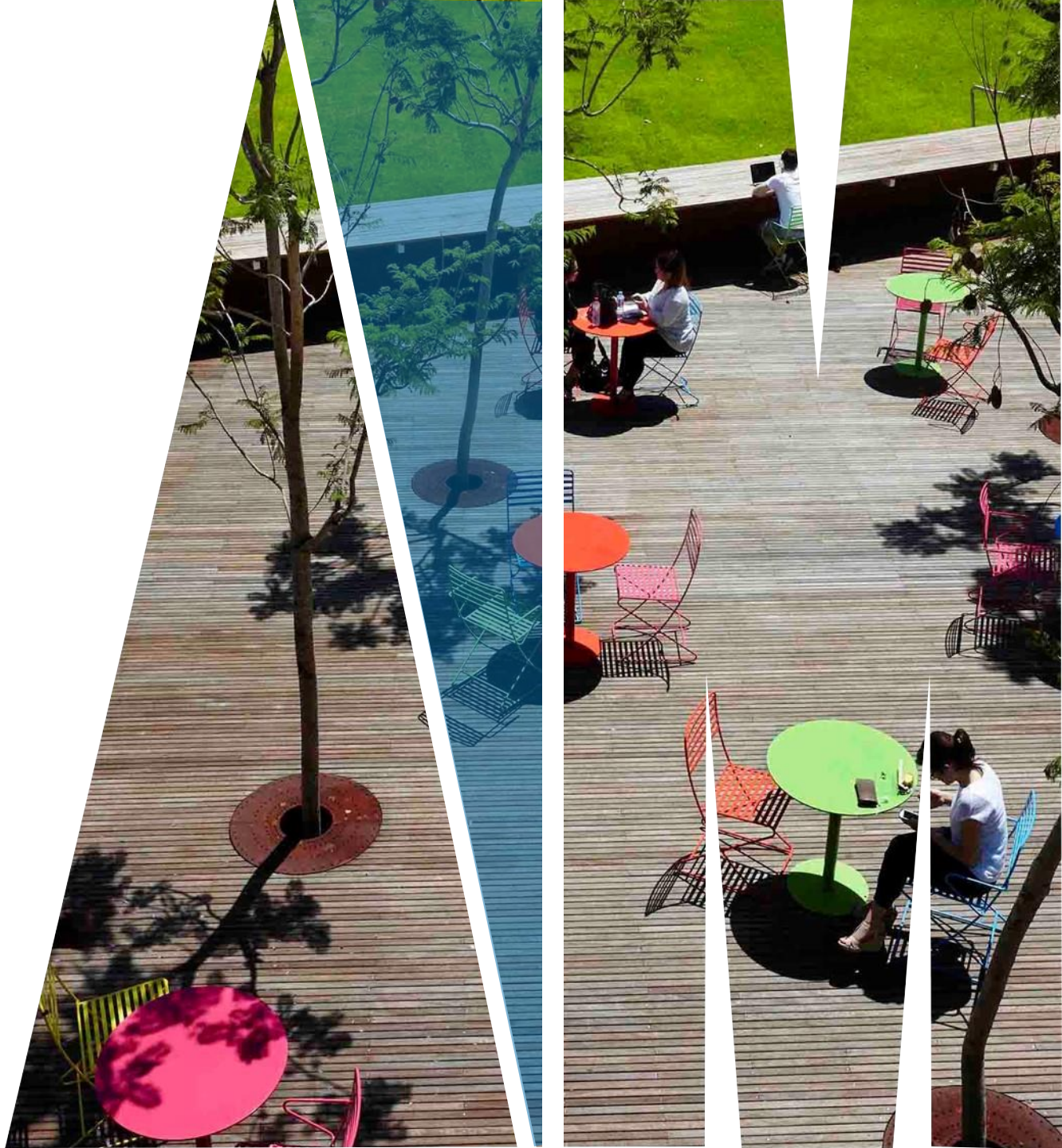
MONASH University

FIT2099 Object-Oriented Design and Implementation

WELCOME!

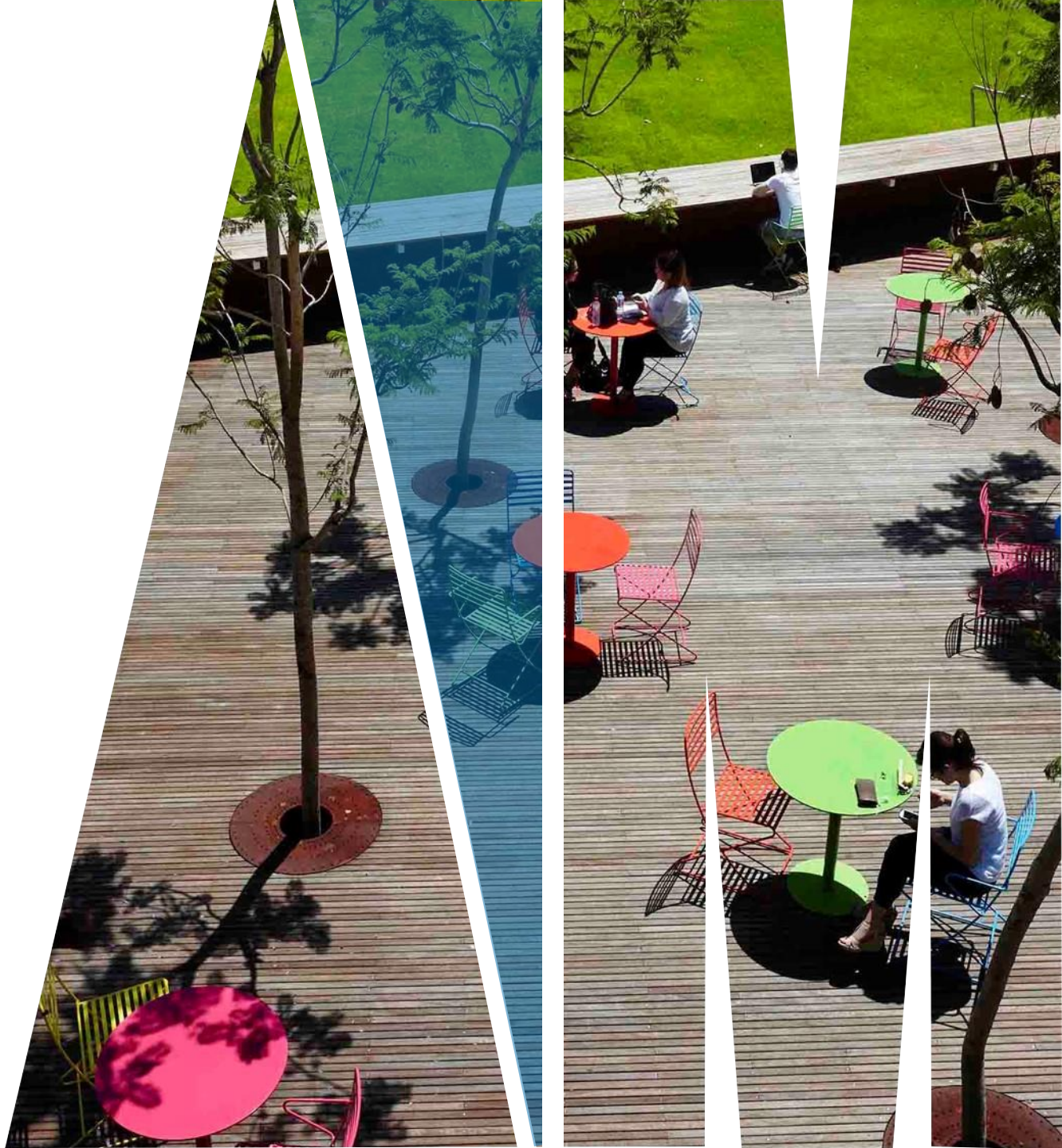


MONASH University

MONASH University

FIT2099 Object-Oriented Design and Implementation

Overview of FIT2099



MONASH University

HOW FIT2099 works

Lectures and **labs**

- Lectures 2 x 1 hour/week (Weeks 1-12)
- Labs 3 hours/week (Weeks 1-12)
- Labs start Week 1, assessed labs in weeks 2-5

Independent study

- **Set readings** on Moodle (these are required)
- **EdLessons** (Weeks 0-5, weeks 1-5 required)

THE LECTURES

Short (video) lessons

– A lecturer will go through:
  - Key design foundations, concepts and principles
  - A coded example that shows how design of a simple system evolves as new features are added and the system becomes more flexible

Suggestions:

Follow along on your laptop for extra hands-on Java practice

It is your responsibility to keep up with lecture materials

A 30 min LIVE lecture Q & A session

(every Tuesday 10.30am AEST)

– A lecturer will be available online:
  - To respond to any questions related to the lecture
  - Provide feedback on any idea or student question

THE

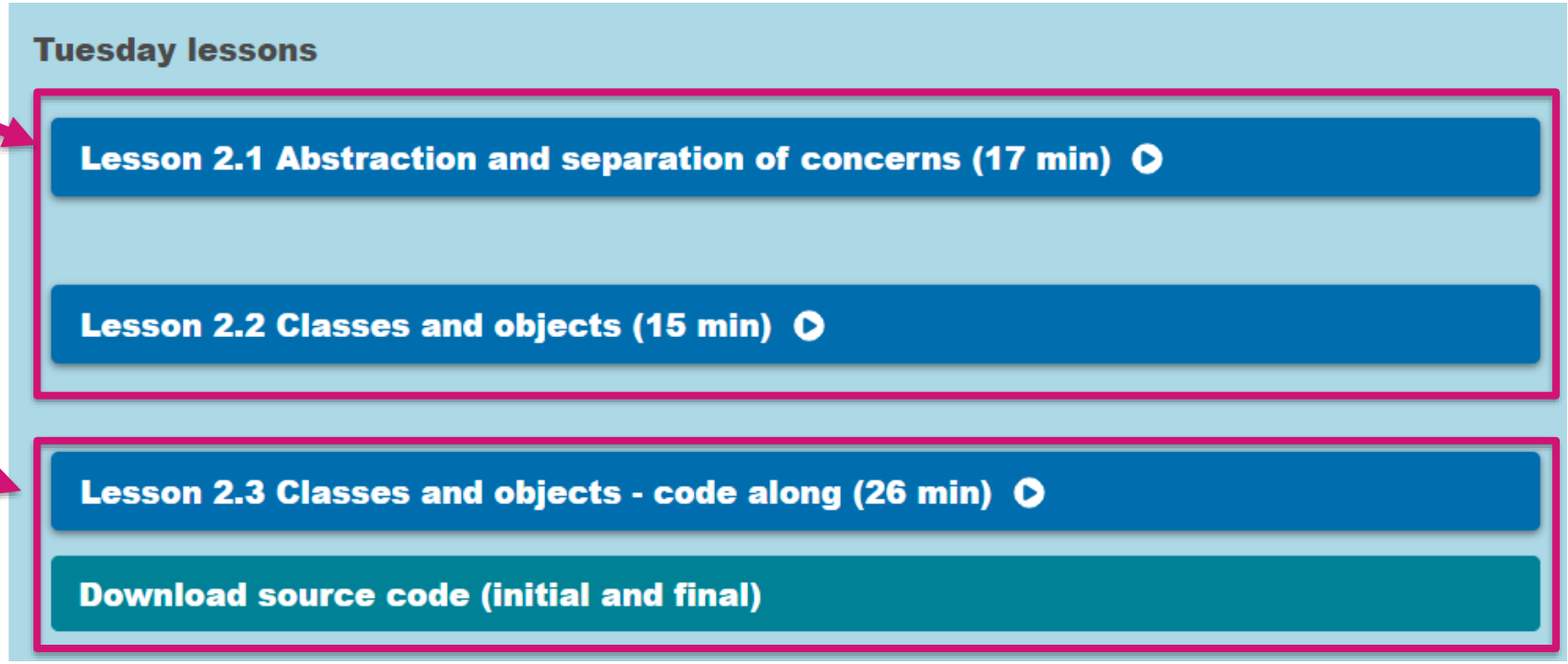
SHORT LESSONS

Week 1

Short Presentations

(theory + design/ implementation examples)

Code along



THE LIVE Q & A

Optional but highly recommended

(every Tuesday 10.30am – 11am AEST)

- It is highly recommended to watch the short (video) lessons 30 min ahead of the regular time.
- The short (video) lessons will be available by Friday a week before
- Please, ask any question regarding the lessons (slides, coded examples, etc).

- A link will be available in Moodle through: Class Streaming

THE ASSESSMENT

Bootcamp held in **labs** and in **EdLessons**, Weeks 1-6 – 10%

- marked in class, done individually
- 3-7 tasks per week in **Weeks 2-5 assessed**

3 Assignments. Done in teams (same for each assignment)

- Assignment 1 – 20%. Due Friday Week 6
- Assignment 2 – 20%. Due Friday Week 8
- Assignment 3 – 20%. Due Friday Week 11

Assignments involve designing and implementing extensions to an existing object-oriented system

Final eExam – 30%

INDIVIDUAL

THE BOOTCAMP

PART A (in the labs weeks 2-5) - 8%

- Weekly Java programming activities which include Object-Oriented principles.
- The cover the first five weeks of the semester during the labs.
- Activities in weeks 2-5 are assessed in the labs.

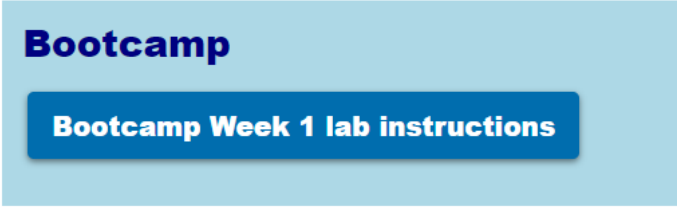
PART B (EdLessons, weeks 1-5) - 2%

- These are intended to help you boost your Java and OO implementation skills in preparation for the Assignments, in addition to the labs.
- They are to be completed online, at your own pace by the end of Week 6.
- You will spend around 1 hour per week depending on your previous experience with Java

INDIVIDUAL

THE BOOTCAMP (in labs, weeks 1-5)

GO TO Moodle – Week 1-5



1. **Attempt the bootcamp** BEFORE the lab
2. **Get feedback** / ask questions to TAs DURING the lab
3. **You can keep working** and updating your repository
4. **Commit** everything before your next lab (a week after)
5. **Go to your handover interview** with your TA in your NEXT lab

(marking and final feedback on **completion** and **quality** of the work)

INDIVIDUAL

THE BOOTCAMP (in EdLessons, weeks 0-5)

Marking

- Based on a **rubric for each week**
- The rubric considers **completeness**, **quality** of work and the **handover interview**
- The rubric will be available via Moodle
- **Two marks** per Bootcamp Part A

INDIVIDUAL

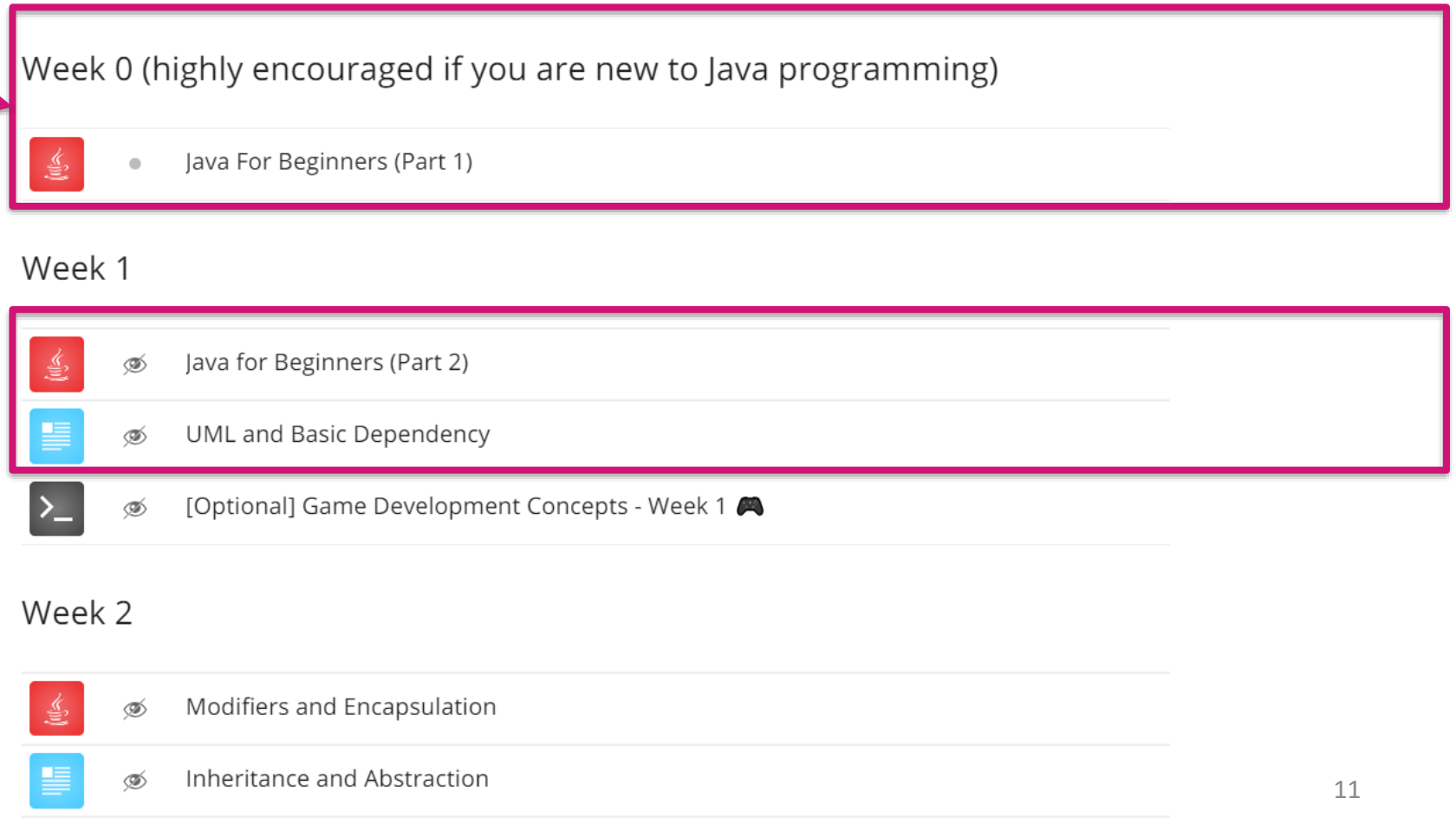
THE BOOTCAMP (in EdLessons, weeks 0-5)

Week 0 is optional

(recommended if you are new to Java programming)

Assessed – completion only

GO TO Moodle – Assessments



INDIVIDUAL

THE BOOTCAMP (in EdLessons, weeks 0-5)

Marking

Completion % by Week 6 x 2 marks = Mark for Bootcamp Part B

For example:

75% x 2 marks = 1.5 marks

TEAM-BASED

THE PROJECT (Assignments 1, 2 and 3)

Project (in labs, Weeks 5-12)

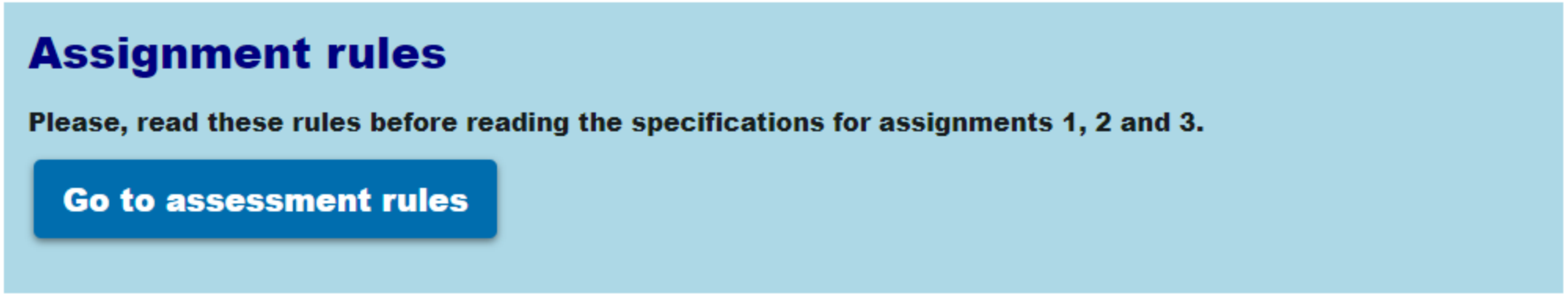
- Done in teams to give you practice at communicating with your peers about design
- It is one project split into three submission points
- The idea is to keep the workload steady rather than have a mad rush at the end
- The **Design** is at least as important as **Implementation**
  - even if you get it 100% working, you can still fail
  - for a good mark your code must be maintainable, extensible, and exhibit other signs of good OO design practice
- Several feedback opportunities
- All the team members are responsible for the whole project, rather than individual parts

TEAM-BASED

THE PROJECT (Assignments 1, 2 and 3)

Please, read the **assignment rules document** for more details

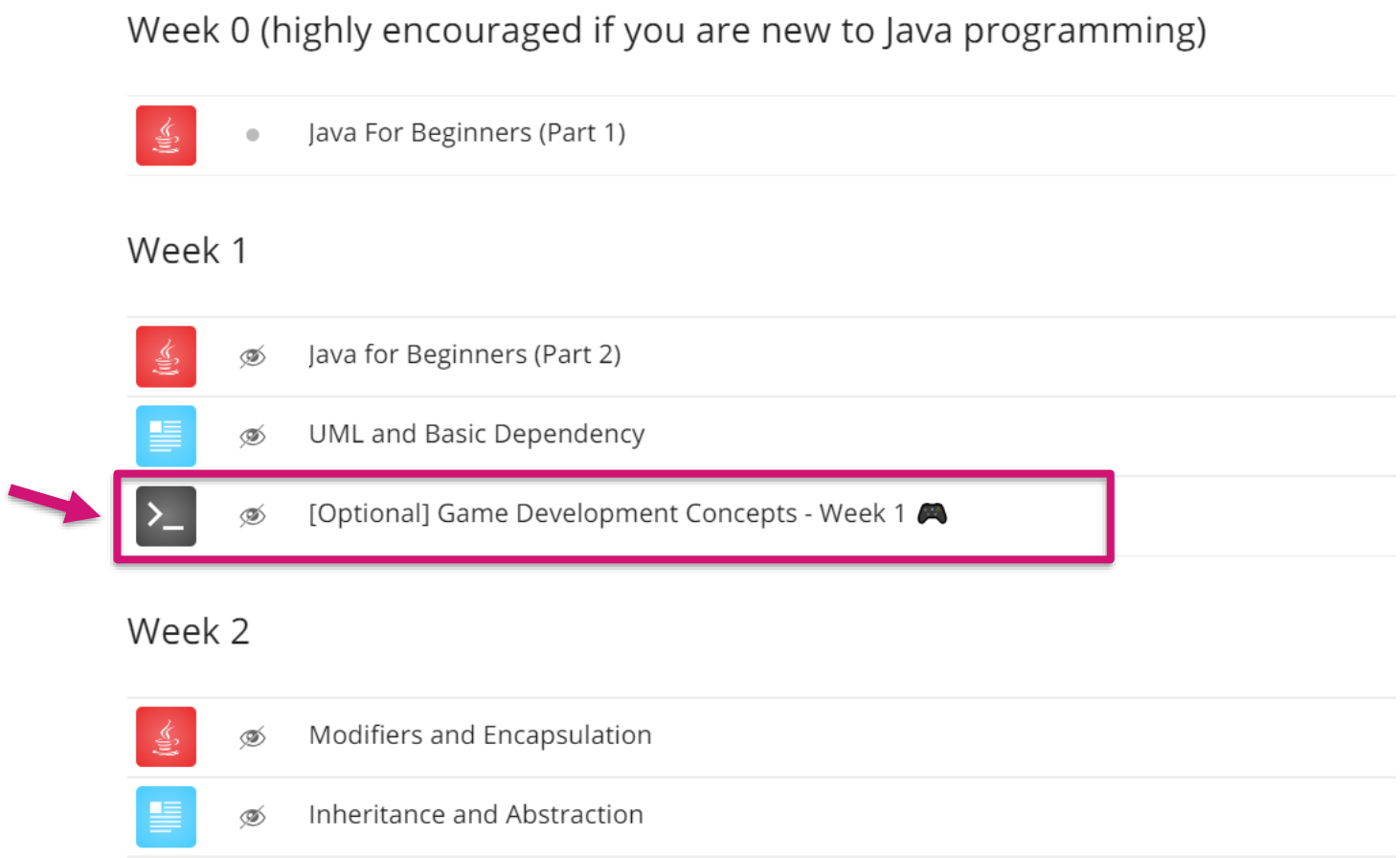
GO TO Moodle – Assessments



ADDITIONAL OPTIONAL SUPPORT FOR THE PROJECT

GO TO Moodle – Assessments

Not Assessed – just to get familiarised with the concepts and the base code to be used in the assignments



INDIVIDUAL

THE FINAL EXAM

- Closed book (you will not need to memorise much)
- **Practical** (design and implementation exercises)
- Example exams will be made available via Moodle
- One live-lecture/QA session in Week 12 focused on the Exam
- **Thirty marks** in total

TEAM-BASED

THE PROJECT (Assignments 1, 2 and 3)

Marking

- Based on a **rubric for each assignment**
- The rubric considers both **completeness and quality** of work in terms of Design AND/OR Implementation
- The **handover interview** is part of the assignment (the week after the deadline)
- The rubrics will be available via Moodle
- **Twenty marks** per Assignment

WHAT SOFTWARE WILL BE NEEDED?

You need a working Java development environment to work on labs and assignments at home

- we suggest JDK15; links are available on Moodle

You will use a git repository to manage all project data for the assignments

- An individual repository and a team repository will be assigned to each of you and your team for the labs and assignments, respectively.
- You will need a git client
- Most modern IDEs have one integrated (including **IntelliJ** which we will be using and supporting int his unit)
- if you learned GitKraken in ENG1003 and want to use it, we won't stop you (but we probably won't be able to support you if it breaks)

STUDY
RESOURCES

There is no set textbook

We will put many readings on Moodle and in EdLessons

- these are **examinable** unless otherwise stated

If you need additional resources on Java programming, there are many, many textbooks and online resources

- post on the **EdDiscussion** forum if you find one that's particularly helpful

GETTING FEEDBACK

- **In the labs**, ask your demonstrator
  - Oral feedback and written feedback (along with the summative assessment)
  - If you're at Clayton, there are two per class, you're encouraged to ask either one at any time
  - You will get direct feedback during each handover interview
- Feedback in **ED Discussion Forum**
  - If you ask something you may get feedback from other students and teaching staff
  - Questions sent by email that are not of a personal nature, but about the unit content in general, will be redirected to Ed Discussion
- Come to a **consultation session**
  - these will be organized when availabilities are known and when demand becomes clear, from Week 2.
  - You can attend ANY consultation session! Bring your questions or assignment drafts.
- Come to the **lecture QA**
  - You can ask questions to the lecturer in turn about the Lecture-related lessons you have previously watched or followed by coding along

THE NEXT FEW WEEKS

Next few weeks in lectures, we will cover:

- fundamental OO concepts, and how they are implemented in Java
- fundamental good coding and design principles and practices
- fundamental OO design principles
- refactoring to improve design
- UML basics: class diagrams, sequence diagrams

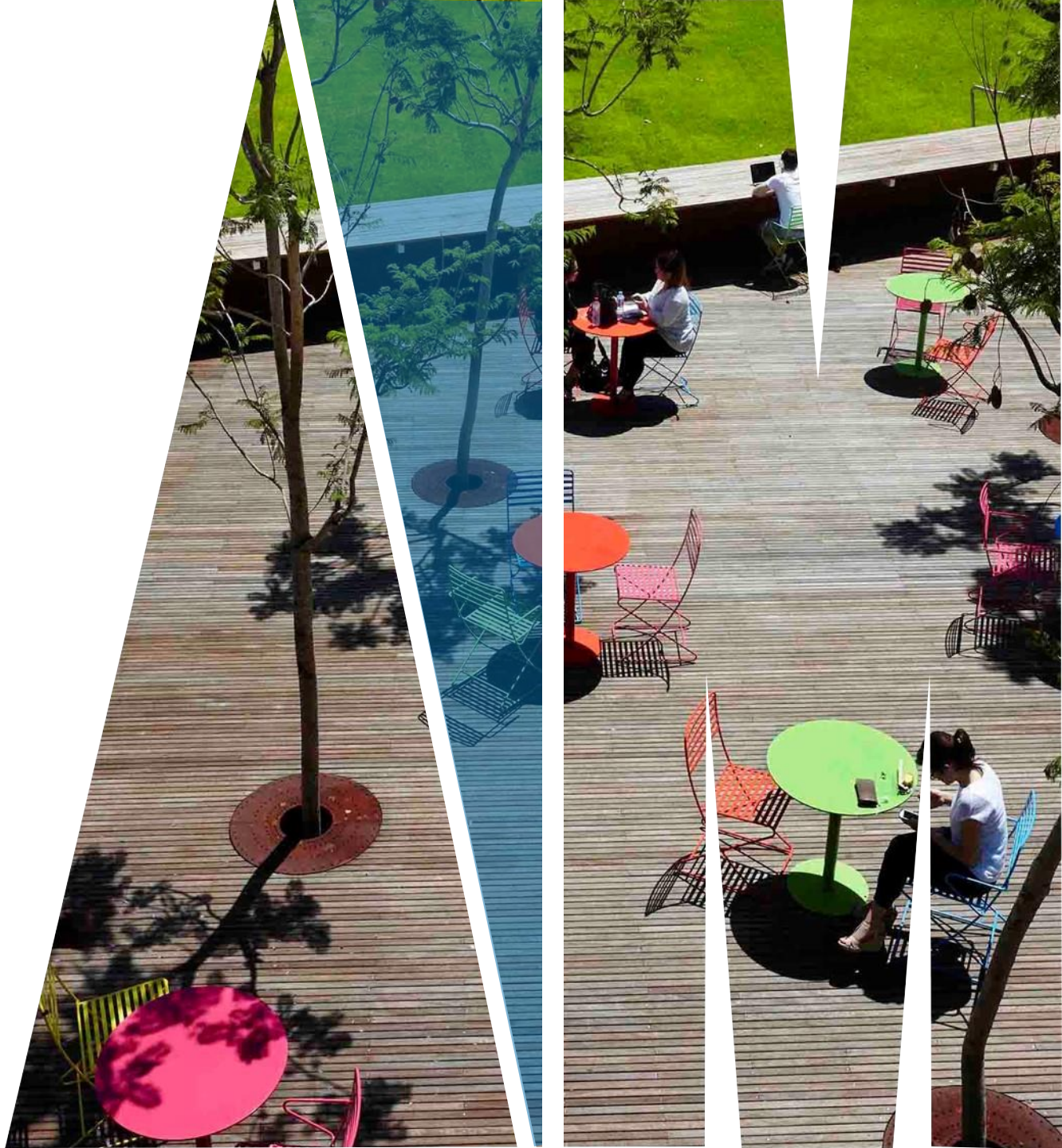
You will:

- Read the readings
- Complete the Java Bootcamp Challenges (in the labs)
- Go through the EdLessons

MONASH University

FIT2099 Object-Oriented Design and Implementation

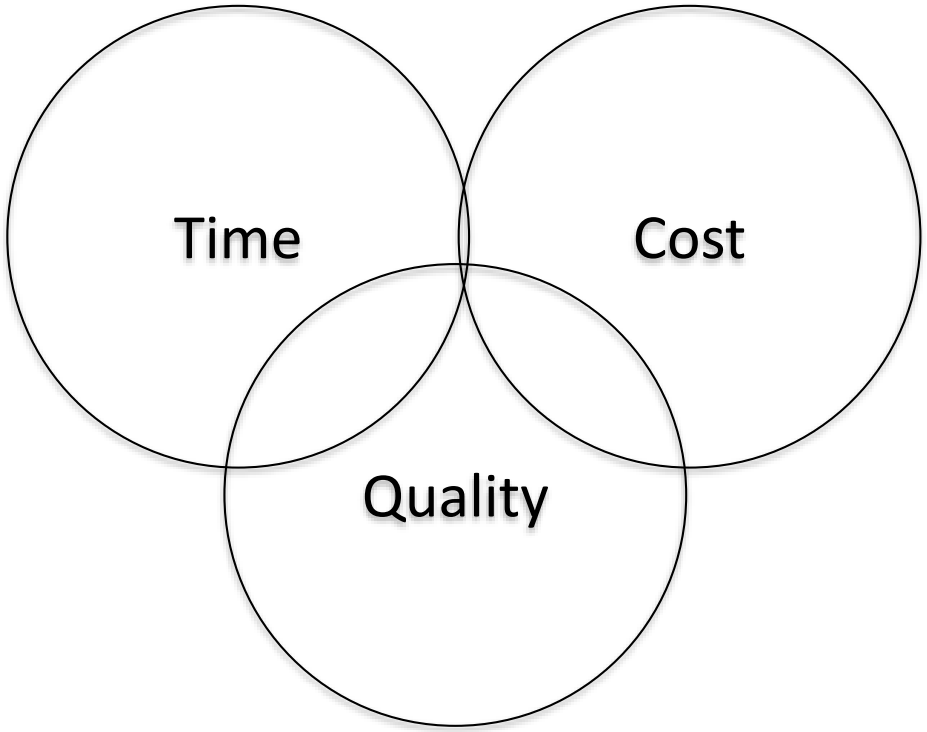
Why object-oriented design?



MONASH University

THE PURPOSE OF THIS UNIT

We want to be able to deliver software on **time**, on **budget**, and of sufficient **quality**



LARGE SOFTWARE PROJECTS CAN EASILY FAIL

Problems with software projects are distressingly common! According to the Standish Group, ***only about a third of software projects ship on time and on budget.***

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011–2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

THE PURPOSE OF THIS UNIT

Software can take a long time to develop

- small, trivial projects might take weeks but large and complex projects take years
- we want to teach you skills that will let you work on **large** and **complex projects**

"The real breakthrough came when we realized that **software is infinite**, while projects are finite. This is the approach to software development that will break the chains that hold back our advances. Make 2020 the end of software projects…"

— Standish Group Chaos Report (2020) Beyond Infinity

SOFTWARE LONGEVITY

Software can and does stay in use for a ***long*** time.

AGENCY	SYSTEM NAME	AGE OF SYSTEM, IN YEARS	AGE OF OLDEST HARDWARE, IN YEARS	SYSTEM CRITICALITY (ACCORDING TO AGENCY)	SECURITY RISK (ACCORDING TO AGENCY)
Department of Defense	System 1	14	3	Moderately high	Moderate
Department of Education	System 2	46	3	High	High
Department of Health and Human Services	System 3	50	Unknown	High	High
Department of Homeland Security	System 4	8 – 11	11	High	High
Department of the Interior	System 5	18	18	High	Moderately high
Department of the Treasury	System 6	51	4	High	Moderately low
Department of Transportation	System 7	35	7	High	Moderately high
Office of Personnel Management	System 8	34	14	High	Moderately low
Small Business Administration	System 9	17	10	High	Moderately high
Social Security Administration	System 10	45	5	High	Moderate

GAO report: https://www.gao.gov/assets/gao-19-471.pdf

SO WHAT?

It’s easy to make small programs that run once

- and you can get away with many, many bad practices in doing so

We want to build software that:

- is **large** (perhaps *millions* of lines of code)
- has an acceptable number of bugs, and **few** other **quality problems**
- **can be fixed** easily when bugs are discovered
- **can be extended** or modified easily when users’ needs change

All delivered within reasonable **time** and **budget**

This is not a solved problem!

WHERE FIT2099 FITS IN?

Design

- making good decisions about how system is put together

Our key focus in FIT2099

Quality Assurance

- checking that artefacts (code and non-code) produced in the process are of satisfactory quality

Touch on in 2099; main focus of FIT2107

Management/Process

- making these and other essential activities happen in a team, at the right time.

Touch on in 2099; main focus of FIT2101

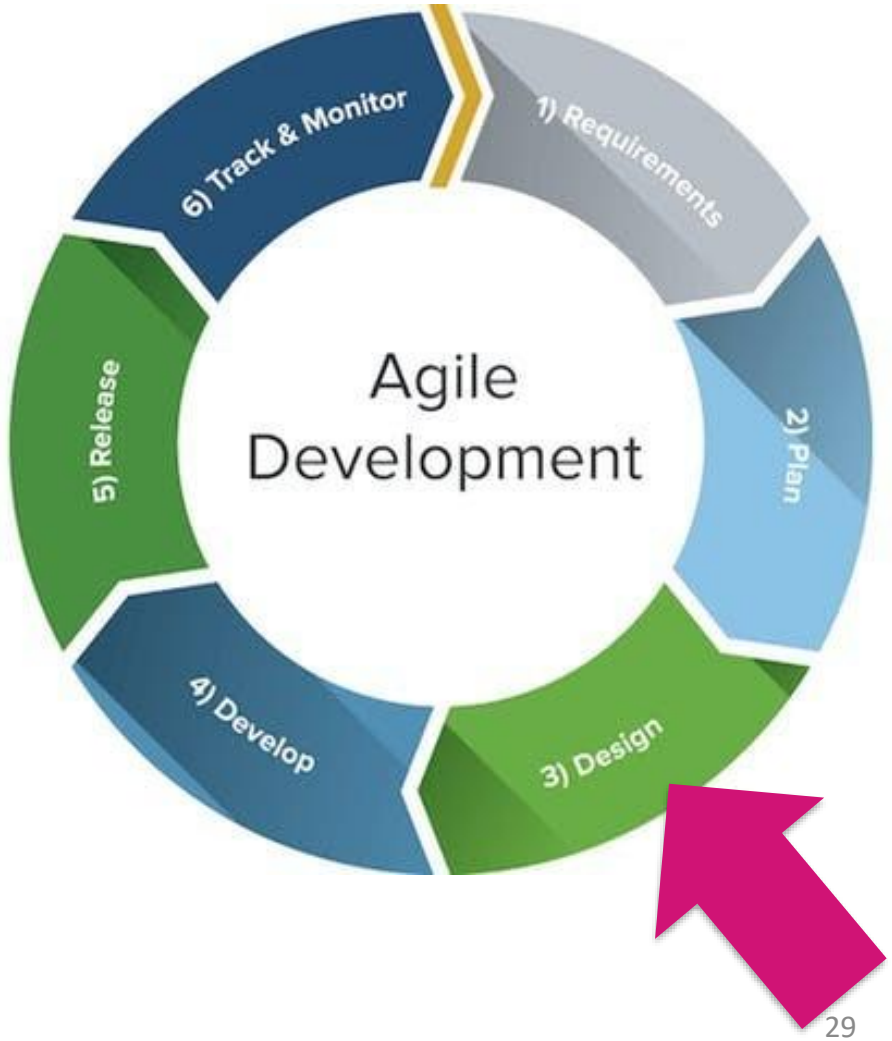
WHAT IS DESIGN?

Design is NOT the production of a “design document” in the “design phase”

Design IS actually the process of making decisions about how the software is to be implemented so as to have all the desired qualities

- of which functionality is only one
- maintainability, extensibility also very important

Design is distinct from software requirements analysis



WHY IS DESIGN IMPORTANT?

You don’t need to put much effort into design if you’re going to build something small and low-stakes, like this doghouse…

…but we want to help you develop the skills to enable you to build the software equivalent of skyscrapers and bridges – large, expensive software projects, with key requirements for performance, security, stability, and many other aspects.

WHY THEN OBJECT ORIENTED-PROGRAMMING?

OOP Ideas developed in ‘60s and ‘70s

- Simula 67 – Ole-Juhan Dahl and Kristen Nygard
- Smalltalk – Alan Kay and others at Xerox PARC

Inspired by biological systems

Popularized by C++, then Java, C#, etc.

- Nearly all modern languages have capabilities or concepts originating in OO (Python, Javascript, Ruby, Scala, etc.)

Nearly all GUI systems were implemented using OO programming

HOW TO EFFECTIVELY DO OBJECT ORIENTED-PROGRAMMING?

OOP has proven good for constructing **large systems**

Early 1990s: Booch, Jacobsen, Rumbaugh (the Three Amigos) and many others (e.g. Robert Martin) introduced

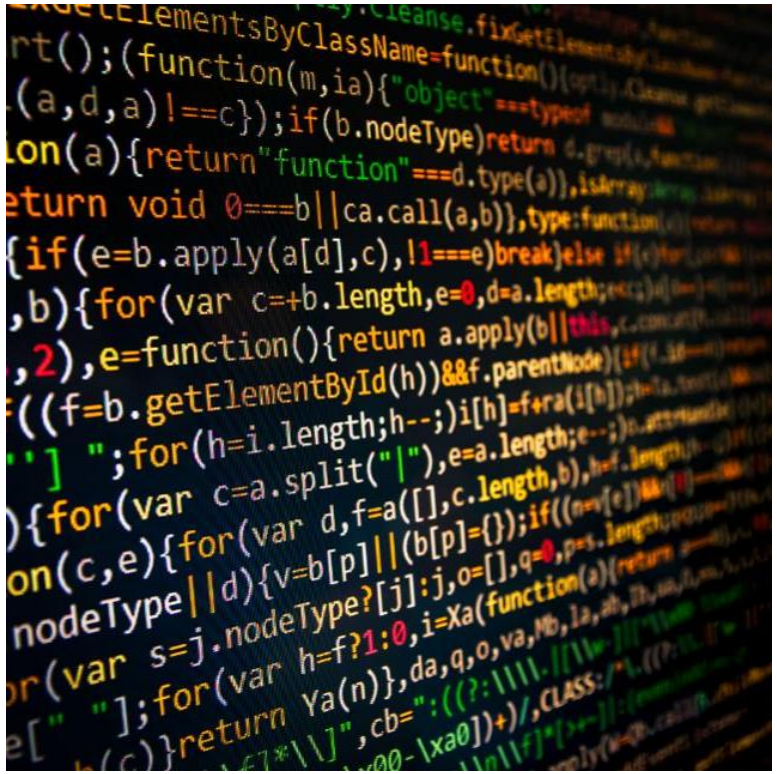
- **notations for modeling object-oriented systems**
  - which eventually evolved into UML (the *Unified* Modeling Language)
- **techniques for identifying the classes, objects, and messages** required
- **heuristics for evaluating** a candidate (or implemented) design for "goodness"
  - design principles, design patterns, design and code "smells", etc.
  - some heuristics from Structured Design were adapted for OO, e.g. coupling, cohesion,

WHAT ABOUT IMPLEMENTATION?

Yes, you will be coding in this unit

The point of design is **to make writing, modifying, and maintaining code easier**

FIT2099 is about the **fundamental principles** of good object-oriented design and implementation, **illustrated** and put into practice using Java

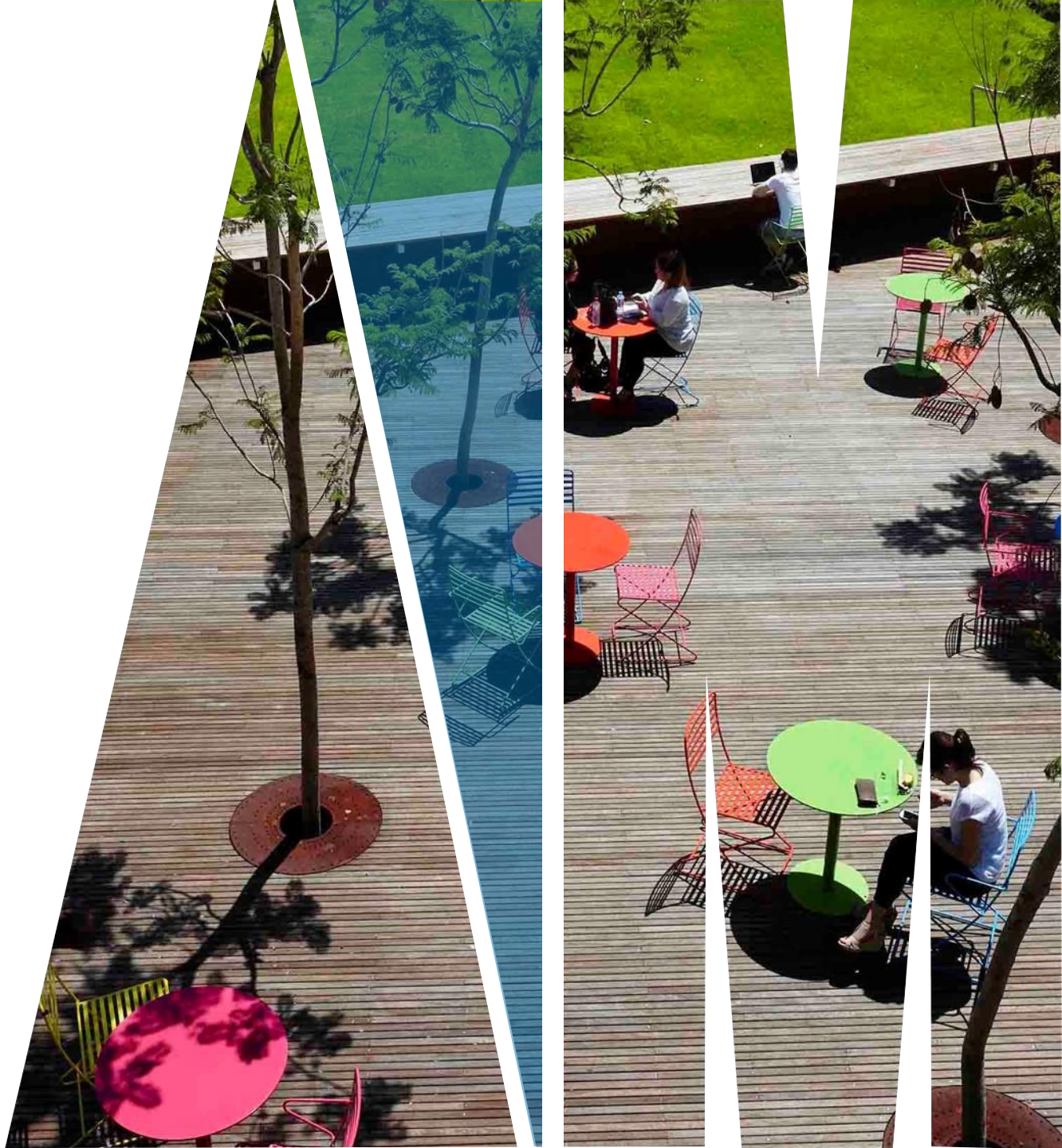


if your assignments aren't well-designed, expect a low mark even if they're fully functional!

MONASH University

FIT2099 Object-Oriented Design and Implementation

Introduction to Object-Oriented Programming (OOP)

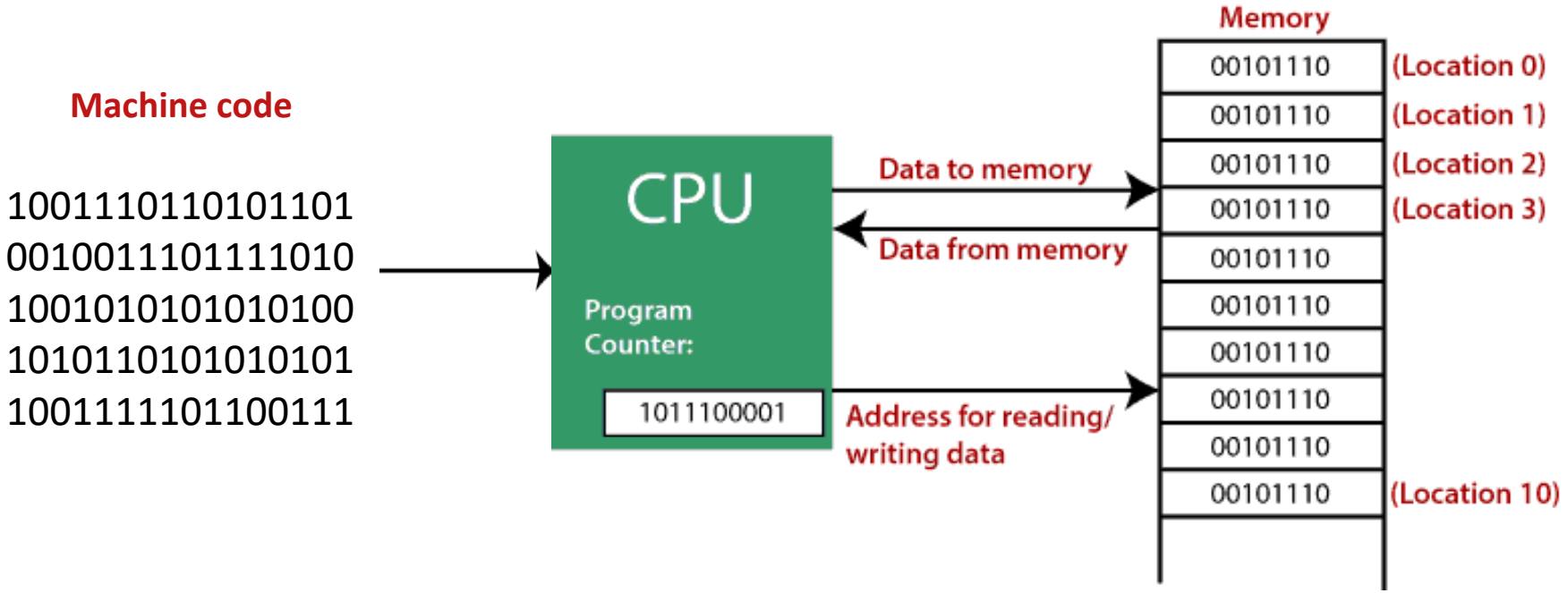


MONASH University

WHAT IS
A COMPUTER PROGRAM?

A computer program is a set of instructions executed by a computer in order perform a particular task

- at the most basic level, this is **machine code** executed by a CPU



HIGHER LEVEL

PROGRAMMING LANGUAGES

Higher level languages use different models to understand and construct programs

There are several programing **paradigms**, e.g.

- imperative
- functional
- procedural
- object-oriented

No matter what higher level language you write in, it still ends up as machine code eventually – but the model helps **programmers** to design and reason about their programs

SOME LIMITATIONS WITH

EARLY, LOW LEVEL LANGUAGES?

Languages such as Assembler, BASIC, FORTRAN

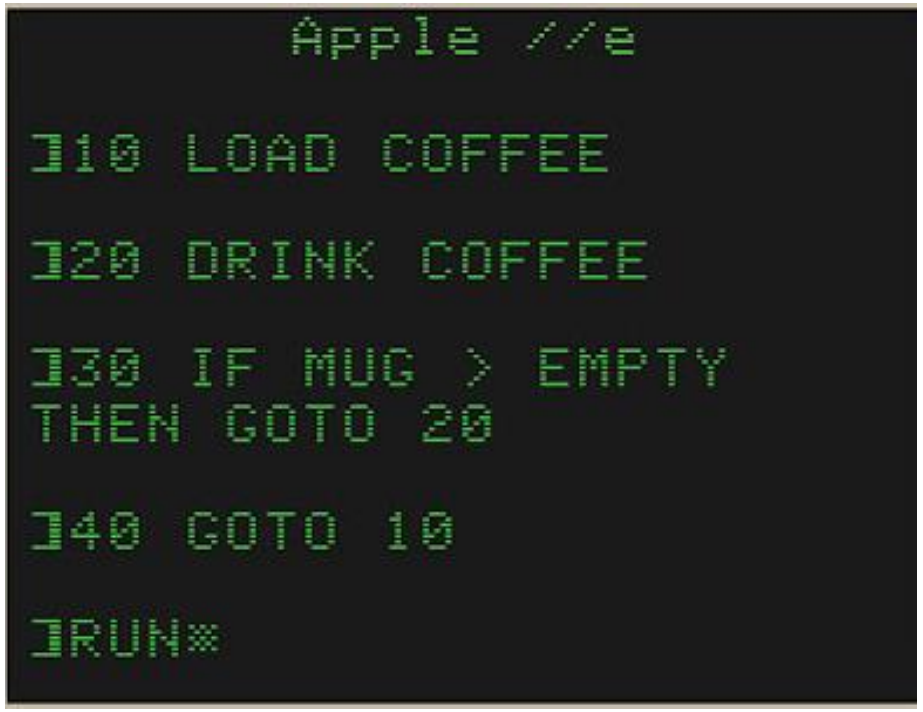
Programs were **unstructured lists of statements**.

GOTO let you jump from any statement to any other statement

All variables had **global scope**

Nightmare to **debug** and **modify**

Not always portable



THERE IS ROOM FOR LOW LEVEL LANGUAGES …BUT…

Programs developed using low level languages are **fast** and **memory efficient.**

Low level languages provide **direct manipulation** of computer registers and storage.

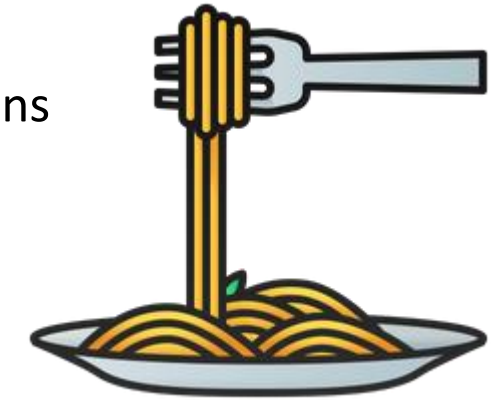
They can directly communicate with **hardware** devices.

BUT IN THE PAST

- programs back then were ridiculously **simple** and **unambitious**
- no such thing as **GUI (graphic user-interface),** networking was rudimentary

Programs were "**write-only**"

- imagine how painful it must have been to work in a team under these conditions



PROGRAMMING PARADIGM:
PROCEDURAL

Key idea: group lines of code into **procedures**

Programs are collections of procedures

- each procedure is a **little mini-program**

Each procedure has an ***interface*** (inputs and return values).

- should be able to change implementation without changing interface

May support defining **data types**

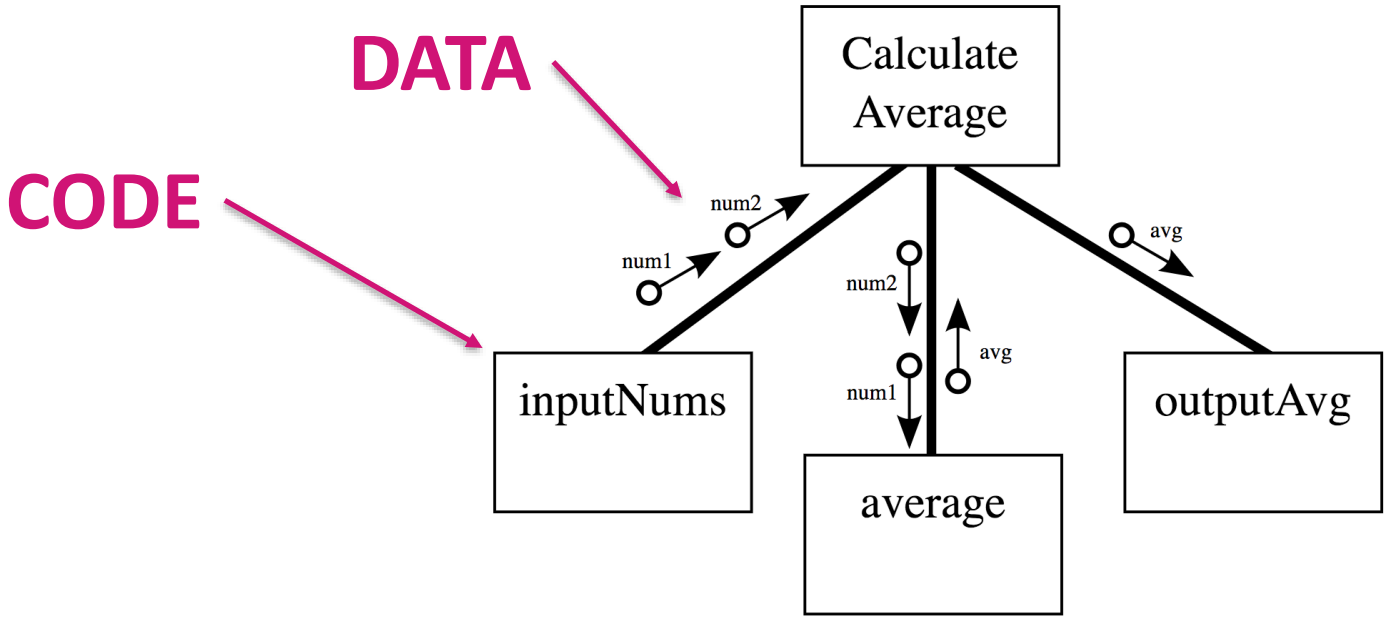
Makes it much **easier** to write/test/debug/extend programs **than "spaghetti" code**

```
        VAR
            Day: DayType;               (* Scanner. *)
            Found: boolean;             (* Tell if a match was found. *)
        BEGIN
            Found := FALSE;
            Day := Sun;
            WHILE (Day < BadDay) AND NOT Found DO
                BEGIN
                    IF DayMap[Day] = S THEN
                        Found := TRUE
                    ELSE
                        Day := succ(Day)
                END;

            MapToDay := Day
        END;

    { Read one character, but do not read past the end of line.  Just
      return a space.
        Pre: InFile is open for reading.
        Post: If InFile was at eoln, Ch is set to ' ', and InFile is
            unchanged.  Otherwise, one character is read from InFile to Ch. }
    PROCEDURE ReadOnLine(VAR InFile: TEXT; VAR Ch: Char);
        BEGIN
            IF eoln(InFile) THEN
                Ch := ' '
            ELSE
                read(InFile, Ch)
        END;
```

Pascal program by Thomas Bennett
http://sandbox.mc.edu/~bennet/cs404/doc/pasdex.html

PROGRAMMING PARADIGM:
PROCEDURAL

Programs consist of procedures (**actions**) that pass **data** to each other



Notice that the boxes represent code (procedures) and the **lines represent data**

Structure Chart Example
https://tinyurl.com/y7ysyhdk

DISADVANTAGES WITH PROCEDURAL PROGRAMMING

Now code was easier to write but **maintenance can still a pain**

- **big systems were still hard to build**

Procedural programming makes **action** the primary unit of organization

- **data is secondary**

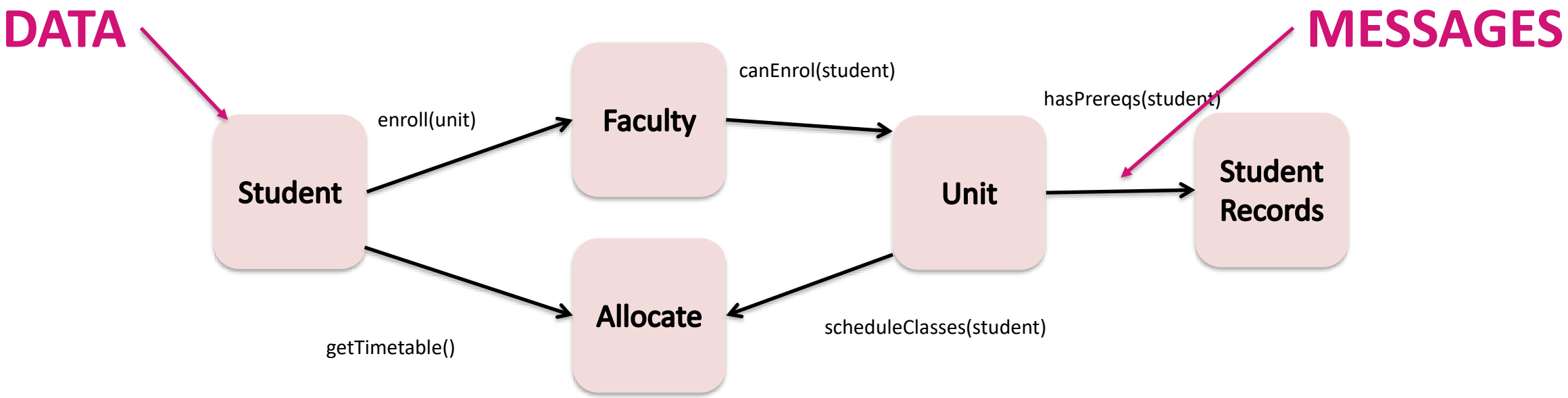
So what happens when the same data structures are needed in many places in the application?

- lots of **repeated** code
- lots of **coupling** (i.e. changes in one function or data structure means you have to change something elsewhere)

PROGRAMMING PARADIGM:
OBJECT ORIENTED

Object-oriented programming **flips this around**

Programs consist of objects (data) that send messages to each other

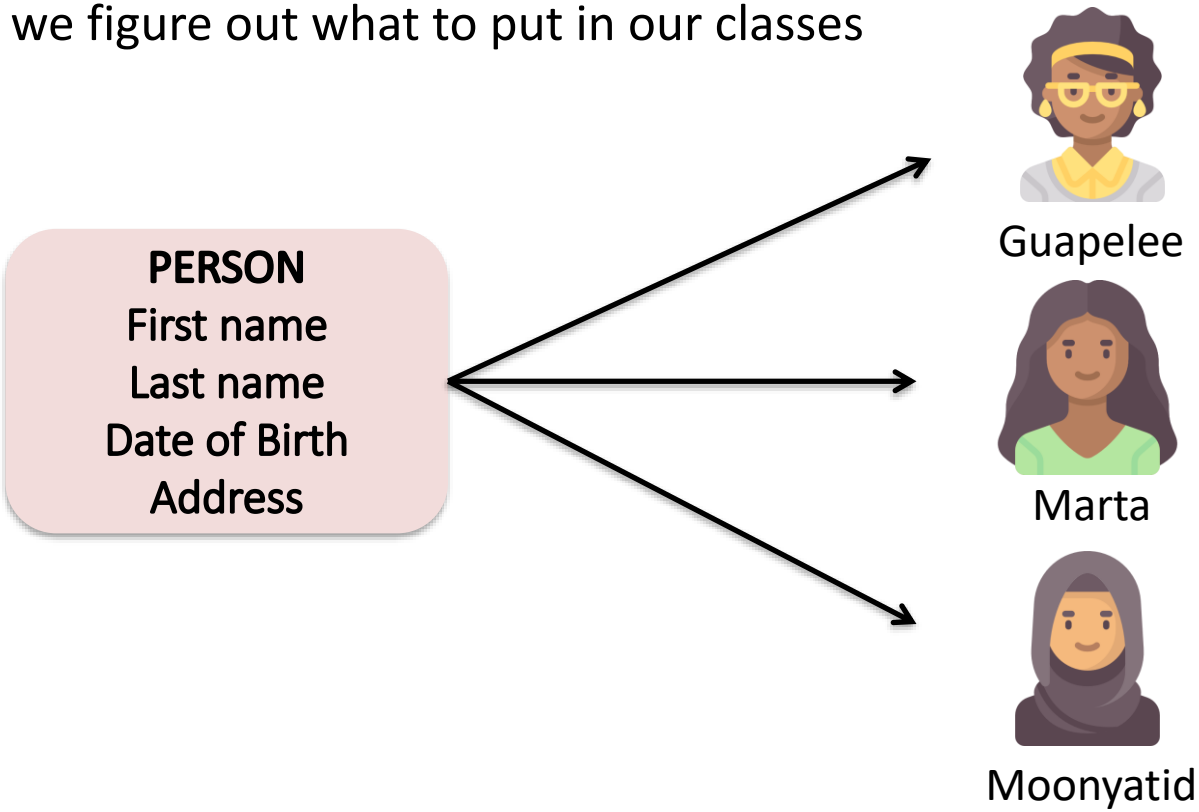


Here, the nodes represent **data** and the **arrows represent code** (method calls)

TWO KEY CONSTRUCTS BEHIND
OBJECT ORIENTED PROGRAMMING

Abstraction: figure out how to represent complex things in simple ways

- by identifying what's important about the thing in the context of the software
- this is how we figure out what to put in our classes



TWO KEY CONSTRUCTS BEHIND
OBJECT ORIENTED PROGRAMMING

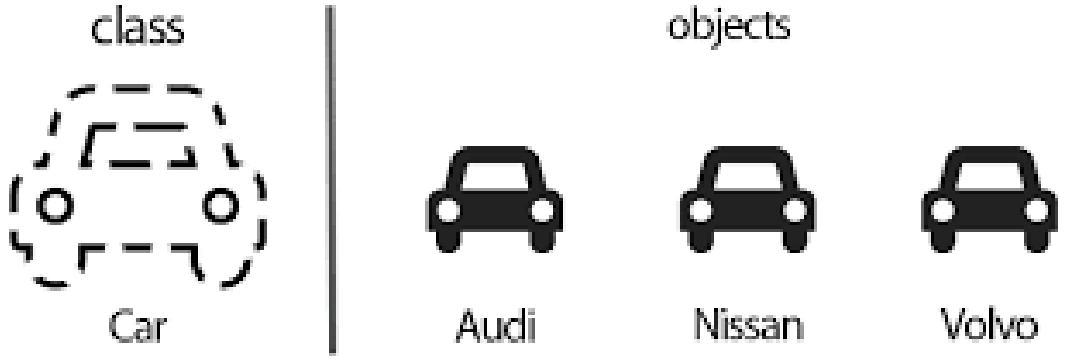
Encapsulation: bundle data that represents an abstraction together with the functions that operate on it

- then **hide implementation details** from other bundles (i.e. classes)
- so you **don't need to think about the implementation**

THE CLASS

A class is an extensible program-code-template for creating objects, providing:

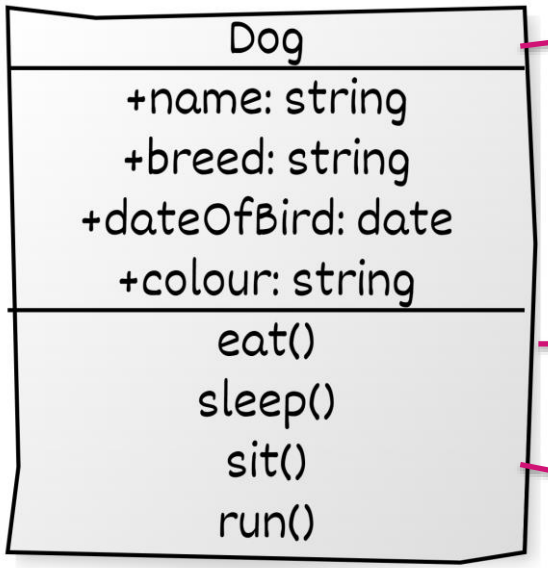
i) initial values for state (attributes) and
ii) implementations of behaviour (member functions or methods)



THE CLASS AS AN ABSTRACT DATA TYPE

Attributes
(data)

Methods
(behaviour)



Name: Buddy
Breed: Boxer
Age: 1 month
Colour: white/brown

Name: Buddy
Breed: Luna
Age: 1 year
Colour: black/white

Name: Dorito
Breed: Chihuahua
Age: 3 years
Colour: light brown

THE OBJECTS

A class is a definition that says what data and methods get bundled together

- e.g. “A Patient has a name, an attending doctor, and a diagnosis, and has a diagnose() method and a bill() method”

An object is an instance of a class

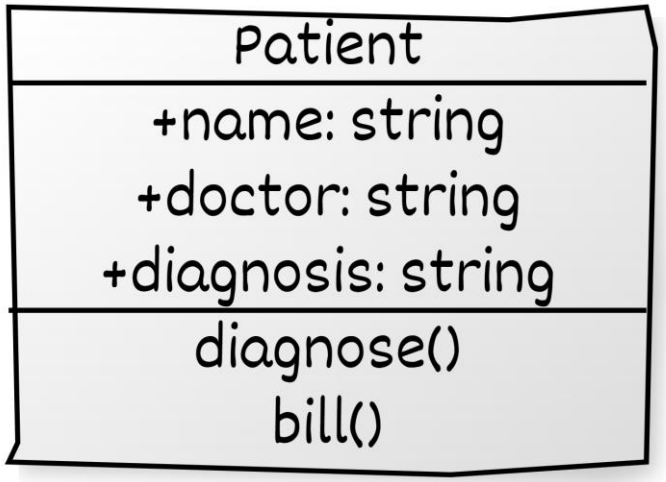
- that is, it’s one specific collection of data
- e.g. “Ravi, Dr Chang, broken leg”
- in Java, all instances of a class share the same method code
  - in JavaScript, you can assign new functions to objects, but you can’t do this in Java



THE MESSAGES (METHODS)

You send a message to an object by **calling one of its methods**

- that is, the collection of methods within a class define which messages its instances can respond to



For example:

```
Patient ravi;    // Ravi is a patient
…
ravi.diagnose(“common cold”);
ravi.bill();
```

Here, we’re creating a Patient called ravi and sending it a “**diagnose**” message and a “**bill**” message

Nearly all of you will have seen this in other units, but you might not have thought of it as **messaging**

ADVANCED CONSTRUCTS

BEHIND OOP

Inheritance: create new classes by using an existing class as a basis

- the subclasses inherit everything the base classes have, and you can add more stuff
- can reuse the features of the base class **without copy/paste**

Polymorphism: send subclass instances a message that's defined in the base class, and they can respond to it in their own specialized way

- yes, this is complicated and hard to explain!

We will dive much more deeply into these concepts in future lectures

- so don't worry if you're confused right now
- the concepts will become clearer in later lectures

BENEFITS OF OOP

A key design principle:

Reduce Dependencies as much as possible

- you will hear it again and again in this unit and others

The corollary is:

If things *must* depend on each other, group them together
(inside an encapsulation boundary – more on that later)

- you will hear this one a lot too

Actions that access or modify a certain data item must depend on that data, and on each other

- so it makes sense to group them together in a class
- this makes it easier to limit the scope of changes to a single class

WHY JAVA?

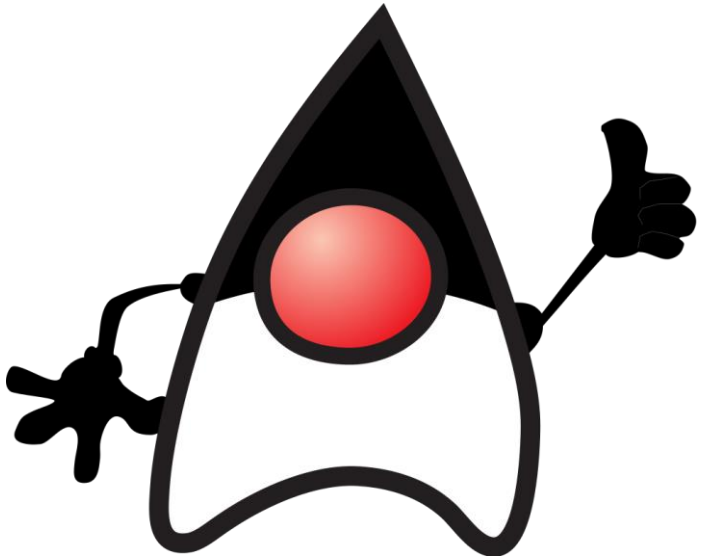
As a teaching language:

- fairly pure OO language
  - keywords match very well with the key OO concepts
    - `extends`, `implements`, `public`, `private`, etc.
- strong, static typing
- memory safety (garbage collection)
- widely used in industry (see tables in last lecture)
- free tools available on all major platforms (MS Windows, MacOS, Linux,…)

As an industry language:

- widely used for enterprise systems
- supports large systems well
- high importance of reliability, maintainability, security

Default language for Android development



Duke, the Java mascot

In fact, Java is one of the top-most used programming languages:

https://statisticstimes.com/tech/top-computer-languages.php

WHY NOT JAVA?

Verbose

- partly inherent to being an industrial-scale OO language
- partly due to design
- tedious to write without IDE support

Harder to use **platform-specific toolkits** (particularly UI)

- if you want to target Windows, C# is easier
- if you want to target MacOS, Objective-C or Swift are easier
- if you want to target browser front-ends, Javascript is easier

Supplied class **libraries not newbie-friendly**

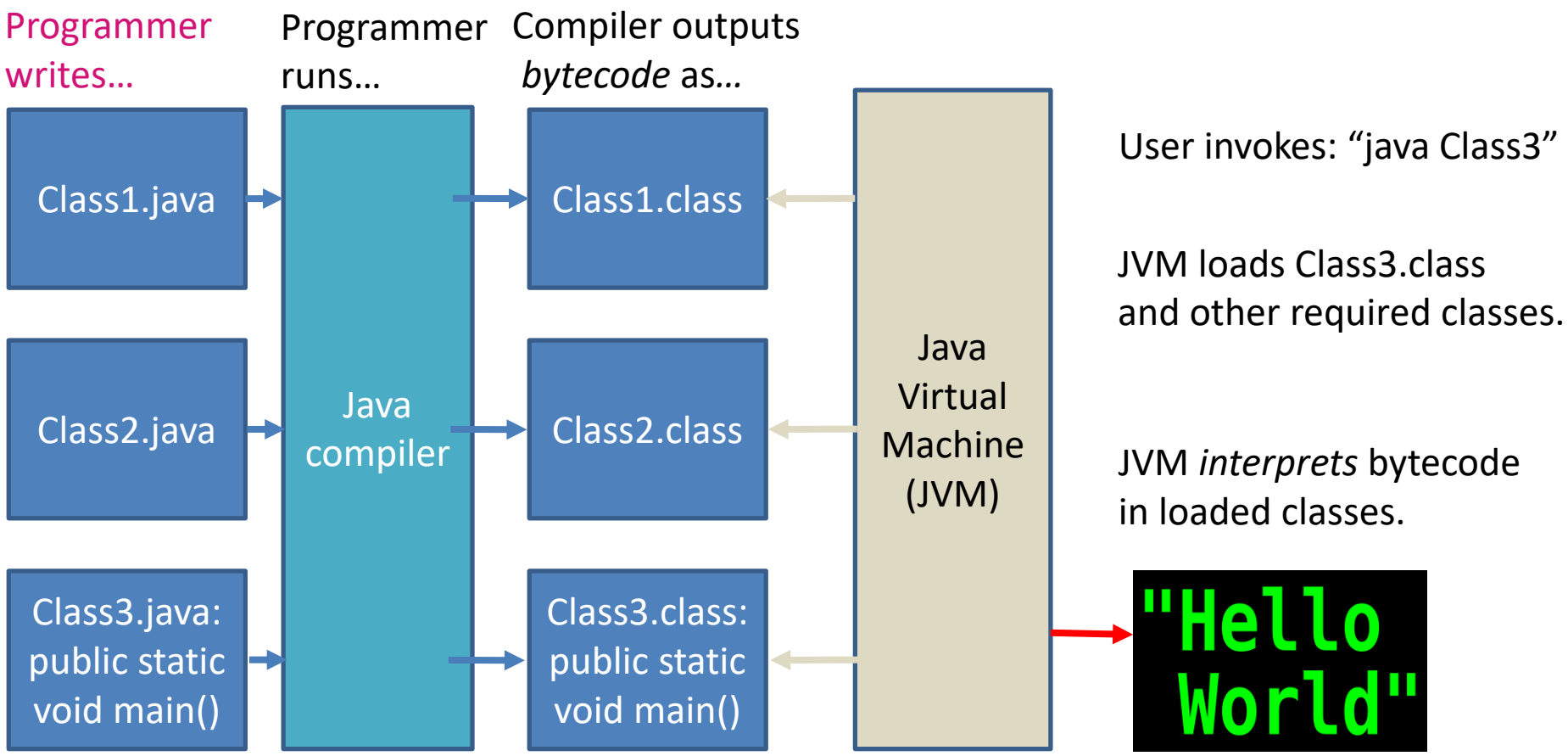
Slower than C/C++ in some circumstance

ALTERNATIVE LANGUAGES

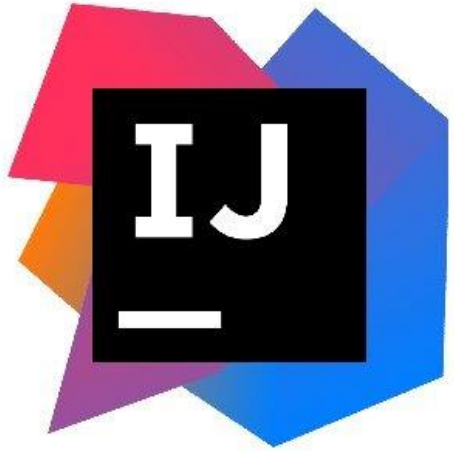
Learning JAVA concepts will help you learn other popular languages

- C# is ***very* similar** to Java
- the concepts you will learn in FIT2099 will also help you learn OO features of **C++, Swift, Python, Ruby**, and many other languages
- the class model of **JavaScript** is very different, but **message passing** works in a similar way

HOW YOUR JAVA PROGRAM WORKS?



THE INTELLIJ IDE



It's possible to

- write Java programs in a text editor
- compile by running the compiler at the command line
- run by invoking the JVM

But this can be tedious

Integrated Development Environments (IDEs) take most of the tedium out

Several IDEs for Java in wide use:

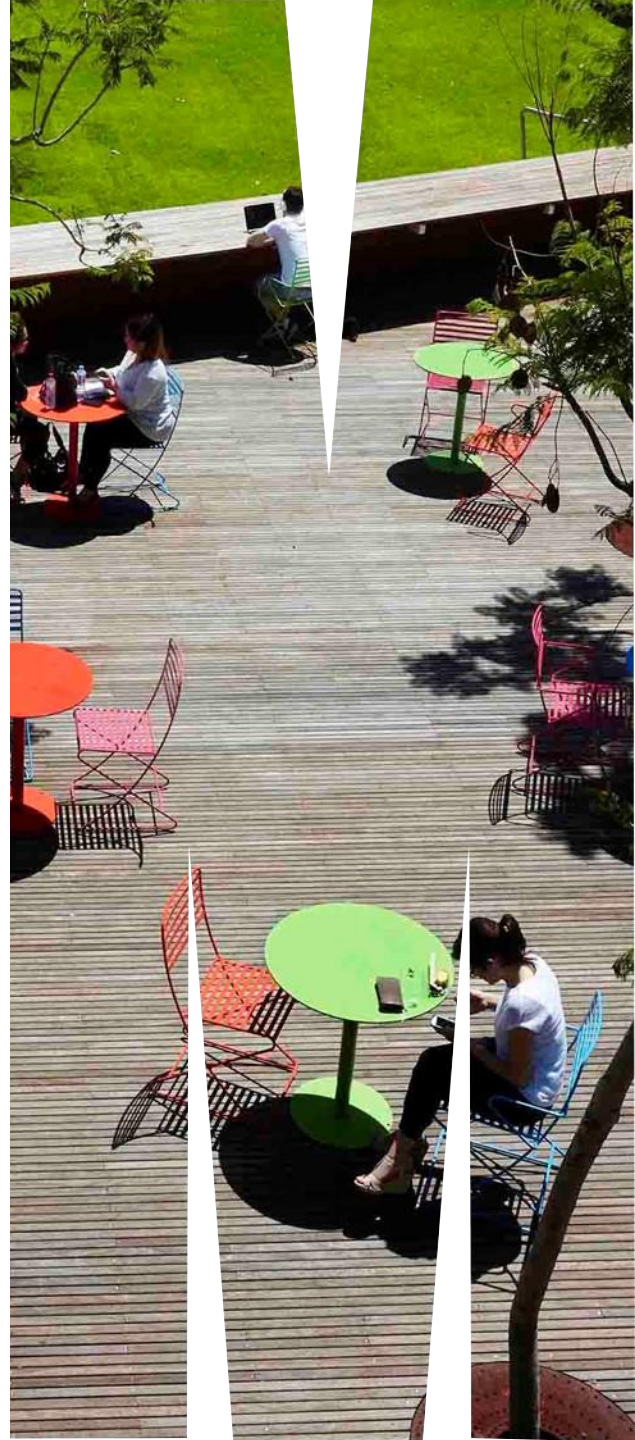
- IntelliJ IDEA https://www.jetbrains.com/idea/
- Eclipse https://www.eclipse.org/ide/
- Netbeans https://netbeans.org/

You can use any tools you like for FIT2099

We will be using and supporting the IntelliJ IDE

MONASH University

Thanks



MONASH University