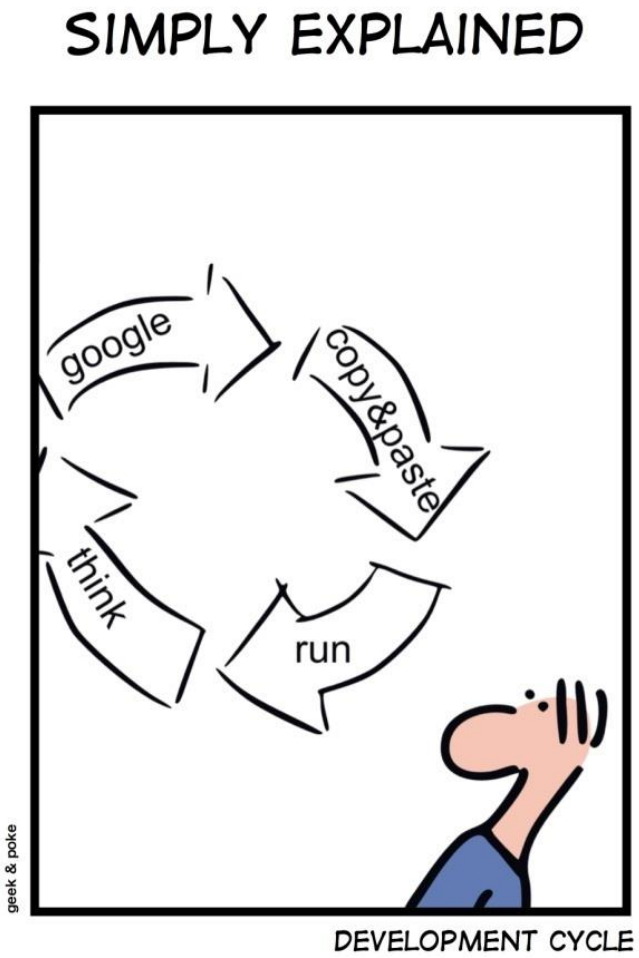


University Mohammed VI Polytechnic

Software Engineering

Pr. Imane Fouad

Imane Fouad

- **Assistant Professor** at UM6P

- **Postdoctoral Researcher** at Inria, Univ Lille
  at **SPIRALS** team in Lille

2018-2021: **PhD** at Inria **Privatics** team, under the supervision of Nataliia Bielova and Arnaud Legout

Main research Interests:

- Web Tracking technologies
- Privacy protection
- Legal compliance

Rules

- No phones allowed.
- **Mute** your phone before entering.
- Keep your phone **out of sight**

Rules

- **Use AI responsibly:** Do not use AI to solve examples, exercises, or assignments.
  Not allowed unless explicitly permitted.

- **Use devices responsibly**: Use your device **only** for class activities.

- Heads up: Misuse (using devices for unrelated tasks) will result in **losing** device privileges during class for the entire semester.

Rules

- Punctuality:
  - Tolerance of 5 minutes for late arrivals.

- Classroom Attendance & Behavior
  - No peer talking or distractions.

Contact

- Send an email to  imane.fouad@um6p.ma
- If you don’t receive a response, send a follow up after 72 hours.

Course Information

- Lectures: 24 hours
- Practical Sessions / Tutorials : 20 hours
- 4h/week over 11 weeks
- Evaluation
  - Midterm exam (30%)
  - Final exam (30%)
  - lab assignments (20%)
  - Final project (20%)

Side note

This course is inspired from Software Engineering - Theory & Practice, S. L. Pfleeger, Introduction to Software Engineering, I. Sommerville SWEBOK v3: IEEE Software Engineering Body of Knowledge, Florian Echtler from Bauhaus-Universität Weimar, ..

Deadline Postponement Token

- 🎟️ **2 Tokens per Semester**
- ⏰ Grants a **3-Day Extension**
- 🔑 **Usable Once Only**
- 🤝 **You must all agree** (choose your own method to decide)
- 🚫 **Not valid for the Final Project Delivery**
- ✅ **Token requires validation**

Course Organization

Discussion

What is software?

What is software?

Computer programs and associated documentation.

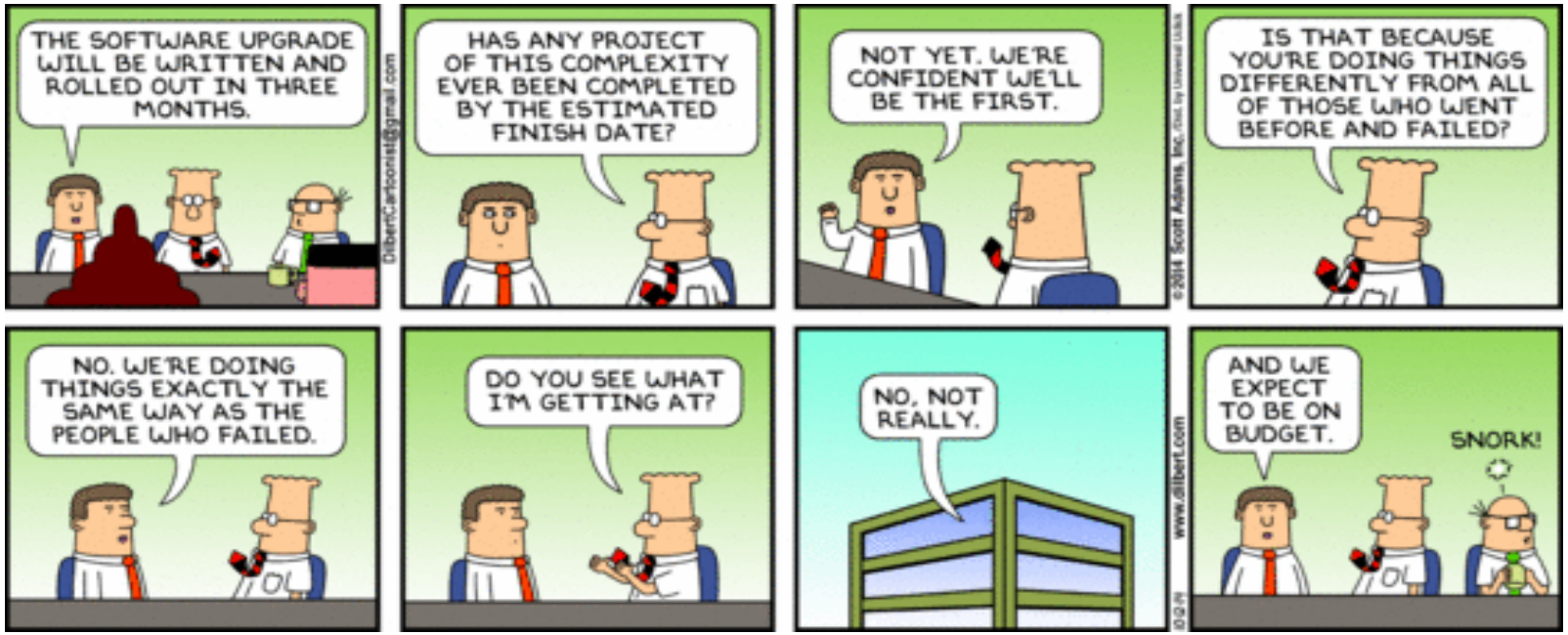
Software products may be developed for a particular customer or may be developed for a general market.

What is engineering ?

What is Engineering?

It is the application of scientific and practical knowledge to invent, design, build, maintain and improve frameworks, process, etc.

What is software engineering?



What is software engineering?

ISO/IEC/IEEE Systems and Software Engineering Vocabulary (SEVOCAB) defines **software engineering** as

"the application of a systematic, disciplined, quantifiable **approach** to the development, operation, and maintenance of software;

that is, **the application of engineering to software**"

Why software engineering?

- Software development is (relatively) easy
- (Unlike building a house)

Image source (CC): https://www.flickr.com/photos/jacobavanzato/16152519186

Why software engineering?

- A first prototype is usually finished quickly …

Image source (CC): https://www.flickr.com/photos/78044378@N00/364003706

Why software engineering?

- … but expanding that prototype is often much more difficult.

Image sources (CC): https://www.flickr.com/.../7165270428, https://commons.wikimedia.org/...Wohnhaus_01.jpg

Why software engineering?

✨ Handling Big Projects

- Software today is huge (think: banking systems, social media, healthcare apps).
- Without structured methods, projects quickly become messy and unmanageable.

💸 To Manage the Cost

- Poorly managed projects overshoot budgets.
- Software engineering helps plan, monitor, and control expenses.

Why software engineering?

⏱️ To Decrease the Time

- Organized processes allow faster development and delivery.
- Teams can work in parallel with clear roles and tools.

🔒 Reliable Software

- Users expect software that works every time.
- Testing, reviews, and standards make software dependable.

🚀 Effectiveness

- Good engineering practices = better performance, easier updates, and longer software life.

Why software engineering?

- Create software that is …
  - ✓ well-structured
  - ✓ easy to maintain
  - ✓ (re-)usable
  - ✓ … and meets its requirements.
- Provide a development process which is …
  - ✓ predictable
  - ✓ adaptable
  - ✓ … and on time.

“A doctor’s mistake is buried in the ground, an architect’s mistake is visible to everyone.”

Famous software failures

- Software is *everywhere*
- → *massive* damage potential

- Billions of € lost

- Hundreds of deaths & injures

Ariane 5 Flight 501

- Left/center: liftoff (June 4th, 1996)
- Right: self-destruct after 39 seconds
- ~ 500 million US-$ damage

Image source (FU): https://www.ima.umn.edu/~arnold/disasters/ariane.html

Ariane 5 Flight 501

- Reasons: integer overflow
  - 64-bit float velocity converted to 16-bit integer
  - Measurement > 32768 → overflow → → uncaught exception → system crash

- Software reused from Ariane 4
  - Old rocket was slower, overflow impossible
  - No overflow handler → system crash with A5

“Morris Worm”

- First “computer virus” in 1998 - by accident
- Research tool to estimate Internet size

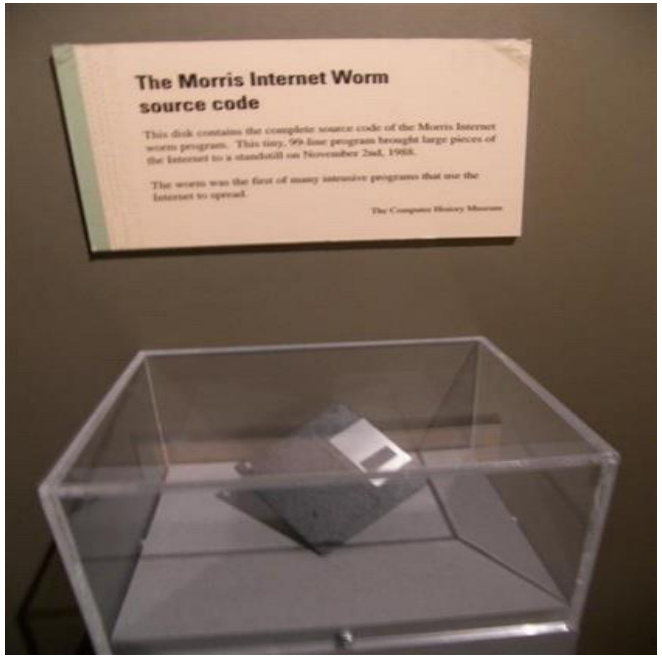


Image source (CC): https://en.wikipedia.org/wiki/Morris_worm#/media/File:Morris_Worm.jpg

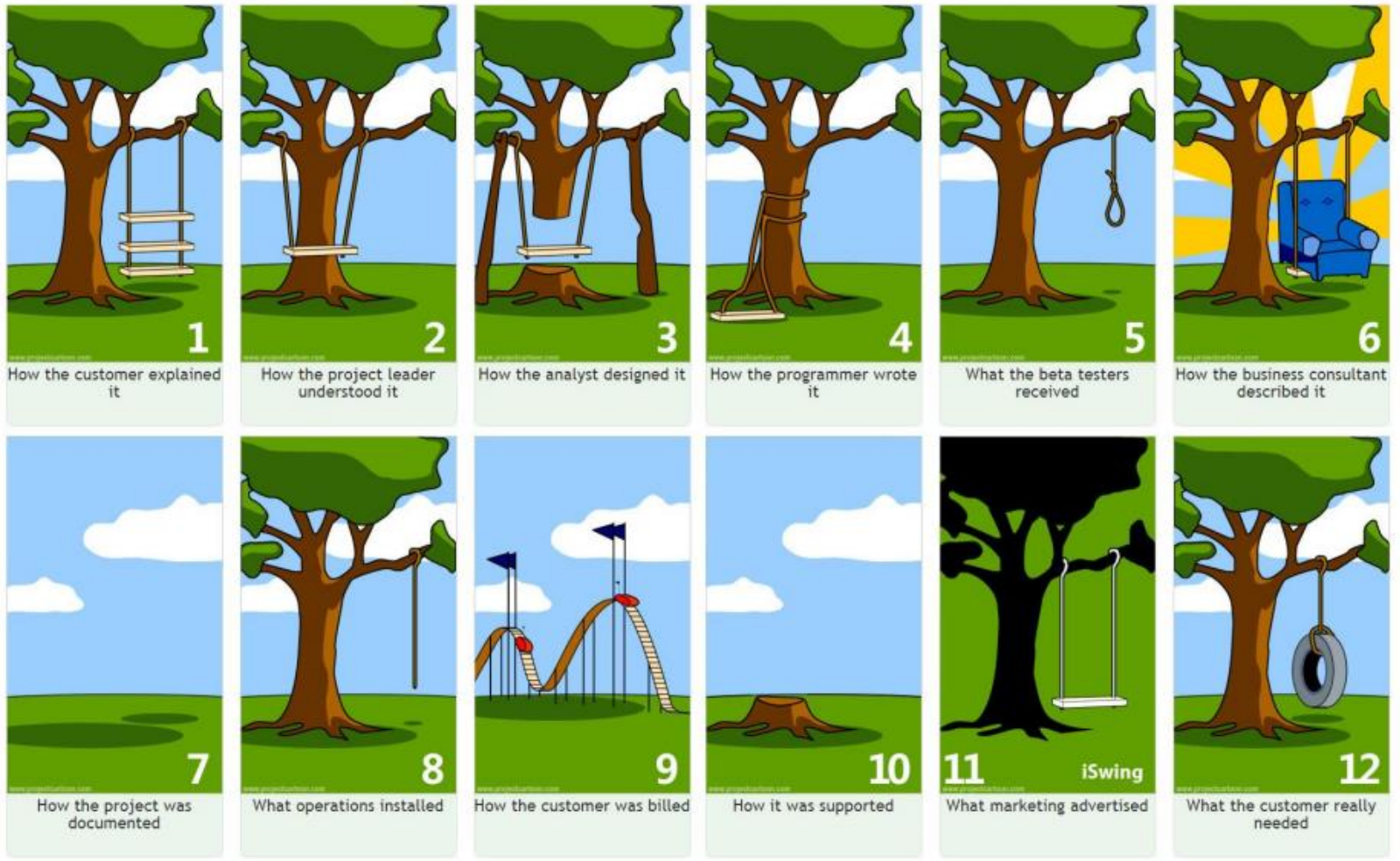
“Morris Worm”

- Heuristic: in 1 of 7 cases, install a second copy of the worm (as anti-removal measure)
- Far too high: most infected systems had dozens of copies running
- Massive slowdown of the entire 1998 Internet

Why Do Software Failures Happen?

Difficulties and Viewpoints

Main reason for failure: **Poorly defined** or **ambiguous** requirements



Challenges in Software Development

Difficult to manage the project and the team.

Clients often struggle to clearly describe their requirements

Requirements and the environment are constantly changing.

Software is intangible (not physically tangible).

Communication gaps between technical and non-technical people.

Hard to detect errors before delivery.

Consequences of Lacking a Methodology

Software that does not meet client expectations.

Slow response times.

Missed deadlines and budget overruns.

Expensive maintenance due to difficulty in managing the software.

This is why Software Engineering (Structured Methodology) became necessary.

Introduction

Discussion

Object Oriented Programming

- Object Oriented Programming (OOP) is one of the most widely used programming paradigm

- Why is it extensively used?
  - Well suited for building trivial and complex applications
  - Allows re-use of code thereby increasing productivity
  - New features can be easily built into the existing code
  - Reduced production cost and maintenance cost

Objects & Classes

- Class: prototype or blueprint from which objects are created

- Object: models a
  - Real world object (ex. computer, book, box)
  - Concept (ex. meeting, interview)
  - …

Objects & Classes

- Class has
  - Set of attributes or properties that describes every object
  - Set of behavior or actions that every object can perform

- Object has
  - Set of data (value for each of its attribute)
  - Set of actions that it can perform

Attributes & Methods

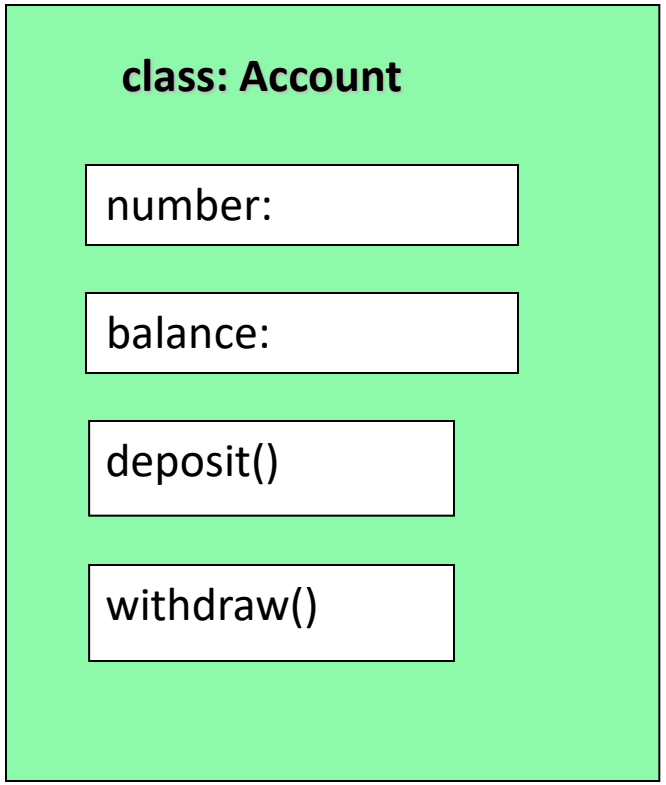
- **Attributes** = the properties or data of an object
- Example: color, speed, brand

- **Methods** = the actions an object can do
- Example: drive(), stop(), honk()

- Together → **define the state and behavior** of the object

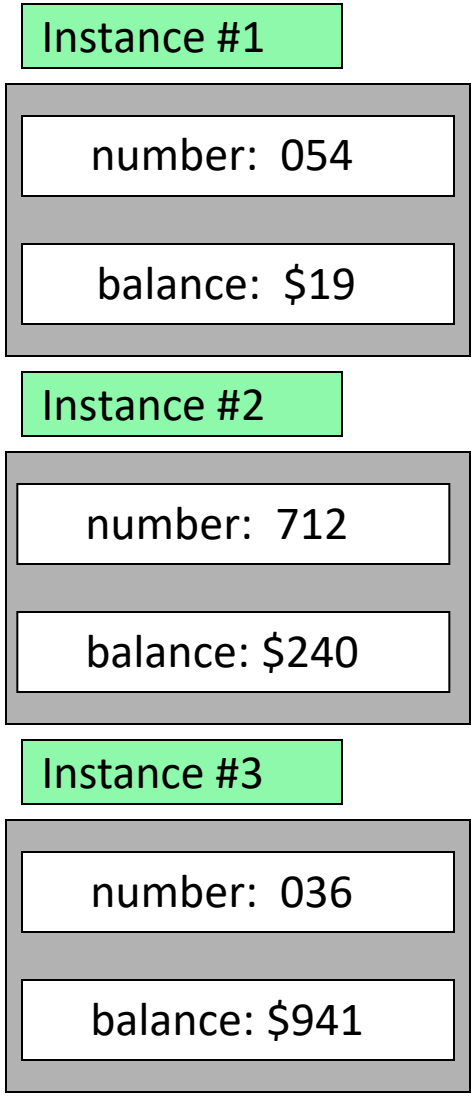
Bank Example

- The “Account" class describes the attributes and behaviors of bank accounts.
- The “Account” class defines two variables (account number and balance) and two methods (deposit and withdraw).



Bank Example - Cont'd

- When the program runs there will be many instances of the account class.
- Each instance will have its own account number and balance (*object state*)



Inheritance

The advantage of making a new class a subclass is that it will *inherit* attributes and methods of its parent class (also called the *superclass*).

- Subclasses extend existing classes in three ways:
  - By defining new (additional) attributes and methods.
  - By overriding (changing the behavior) existing attributes and methods.

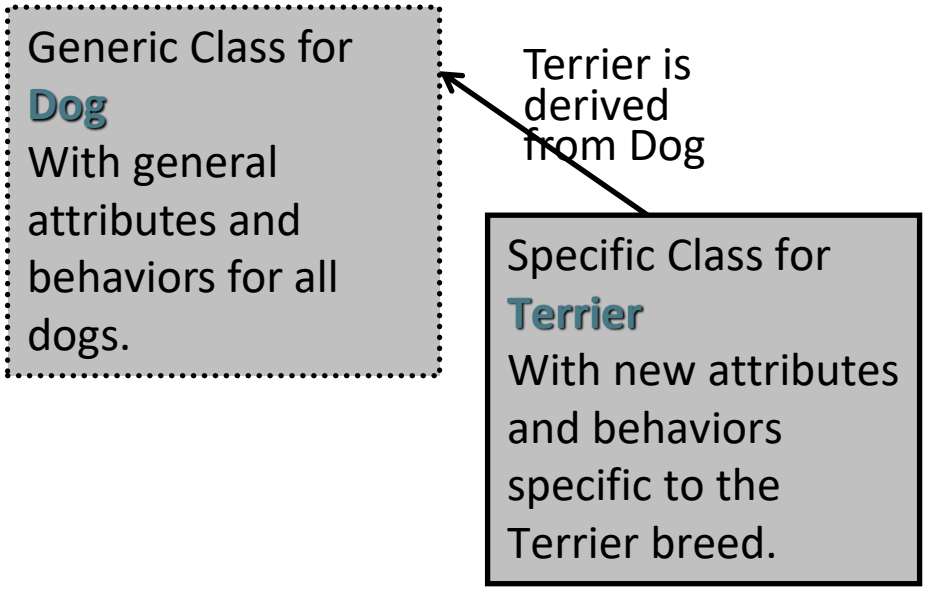
Subclasses

When a new class is developed a programmer can define it to be a ***subclass*** of an existing class.

- Subclasses are used to define special cases, extensions, or other variations from the originally defined class.

Examples:

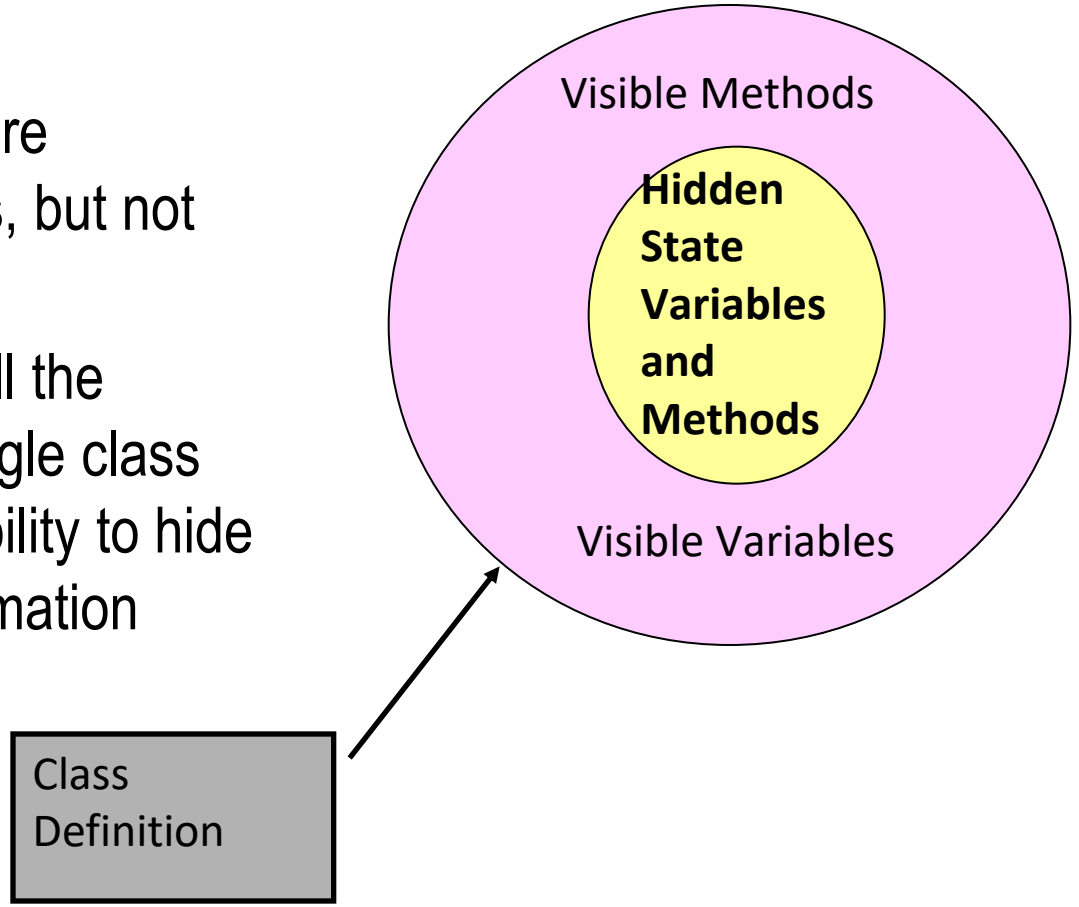
- ***Terrier*** can be defined as a subclass of ***Dog***.



Encapsulation

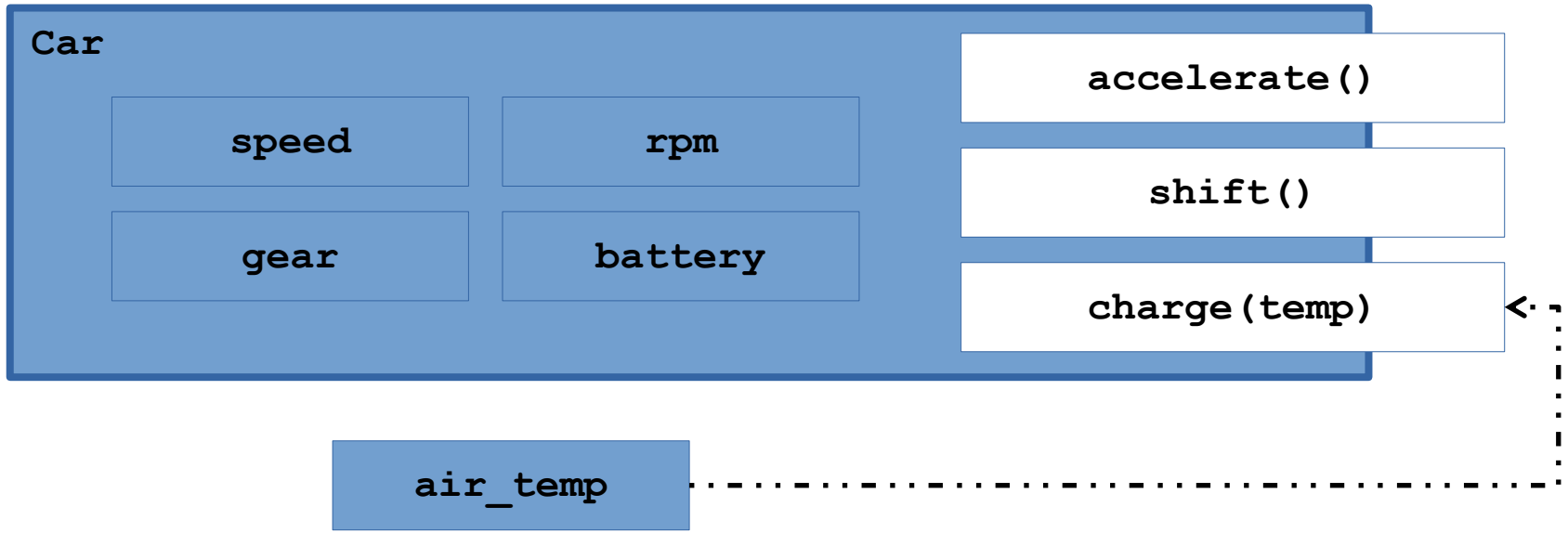
When classes are defined, programmers can specify that certain methods or state variables remain hidden inside the class.

- These variables and methods are accessible from within the class, but not accessible outside it.
- The combination of collecting all the attributes of an object into a single class definition, combined with the ability to hide some definitions and type information within the class, is known as encapsulation.



Encapsulation

- Core idea of OOP: *encapsulate* related data
- Data is no longer directly accessible
- Class provides *methods* to manipulate data



Encapsulation

- Data and methods have *visibility* or *scope*
- Common levels:

–Public: visible/accessible to everyone

–Protected: only visible to subclasses

–Private: only visible to other class members

- Rule of thumb: try to avoid public members (otherwise, encapsulation is sidestepped)

OOP – Key Principles

Key Principles:

- **Encapsulation** – Keep data safe inside objects; access via methods.
- **Abstraction** – Hide unnecessary details; show only what is needed.
- **Inheritance** – Reuse and extend behavior from existing classes.
- **Polymorphism** – Objects can take many forms; same interface, different behavior.

OOP

Discussion

What are the objectives & expected outcomes?

Characteristics of good software

Operational Characteristics	Transitional Characteristics	Maintenance Characteristics
- Budget	- Interoperability	- Modularity
- Usability	- Reusability	- Maintainability
- Efficiency	- Portability	- Flexibility
- Correctness	- Adaptability	- Scalability
- Functionality		
- Safety		
- Security		

What are the objectives & expected outcomes?

Learn how to:

- Specify requirements for a large-scale software system.
- Specify the architecture of the system based on the requirements specification.
- Design and implement the subsystems of the system's architecture.
- Test the system in a principled way that guarantees the quality of the result.
- Organize the delivery of the system.

What are the objectives & expected outcomes?

- ✓ Handling large and complex projects 📊🛠️
- ✓ Managing cost and time effectively ⏰💰
- ✓ Ensuring quality and reliability ✅🔧
- ✓ Effectiveness 🎯⚡

Software Processes

Software Processes

- SPs are “activities involved in producing a software system”
- SP models are “abstract representations of these processes”

- *There is no ideal process.*
- *One size does not fit all.*

Activities in Software Development

- Every software project is made up of multiple activities.
- Each activity involves several people working together.
- An activity has **inputs** and **outputs**. Outputs often include **deliverables**.
- **Deliverables** are products or documents created by an activity

  and used by other activities.
- Examples of deliverables: documents, project plans, source code.

Main Activities

Requirements Analysis

Design

Coding / Implementation

Testing

Maintenance

Requirements Analysis

- Defines **functionality** of/**constrains** on the software product
- Also known as *requirements engineering*
- Nearly always the **initial step**
- Sub-activities:
  - Feasibility study
  - Requirements elicitation/analysis
  - Requirements specification
  - Requirements validation

Requirements Analysis → Challenges

- **Communication Gap:** The client may use a different “language” than the developers.
- **Misunderstandings and Omissions:** Complex requirements can be forgotten or misunderstood.
- **Difficulty in Estimation:** Hard to predict time, cost, or effort accurately.
- **Changing Requirements:** Client’s needs may change during the project.

Design Phase

- Choose **technical solutions** that meet client expectations.
- Help establish a **project plan** based on the chosen solutions.
- Create the **architecture** of the system (overall structure).
- Build **mockups or prototypes** to visualize the solution.
- Apply **technical knowledge** to transform ideas into a concrete plan.

Design Phase → Challenges

- Strong dependency on requirements analysis
- Many possible solutions → choosing the best one is hard
- Requires strong technical skills
- Technology evolves very quickly

Coding Phase

- **Turn design into operational code**
- Uses **programming languages** (e.g., Python, Java, C#…)
- Represents the **largest part of the work** in software projects
- Involves the **largest team** of developers
- Requires a **shared code repository** (location, structure, templates…)

Coding → Challenges

- Project management with **large teams**
- **Integrating code** from many developers
- **Ensuring shared understanding** of the project
- **Standardizing** methods and practices
- Developer **mobility** (when someone leaves or joins mid-project)
- **Different technical skill levels** among developers

Software Testing Basics

- Software **quality**
- **Conformity** to specifications
- Many test types (2 main ones):
  - **Unit tests**: test small pieces (functions, classes)
  - **Functional tests**: test features as a whole
- White-box testing: with access to source code
- Black-box testing: without access to source code

Difficulties in Testing

- Requires **focus** → can be **boring / repetitive**
- Goal: **find as many bugs as possible, as quickly as possible**
- Some parts are **hard to automate**

Main Activities

Requirements Analysis

Design

Coding / Implementation

Testing

Maintenance

Objectives & Expected Outcomes

Discussion

Project

The project will apply all phases of software engineering:

- Requirements Analysis
- Design
- Coding
- Testing

Deliverables at **each stage** will show your progress (documents, diagrams, code, reports).

Project Instructions

- Duration: **1 semester**
- Workload: **~½ day per week**
- Scope: Manageable but complete project lifecycle
- Level: Matches **1st-year engineering class** background
- No advanced skills (e.g., ML)
- Focus on **software engineering process** (not just coding)

Assignment

- **1 Slide only**
- Title = Name of your App
- Content:
  - Short **description** of the app
  - List of its **functionalities** (what it can do)

- Footer: Add *your full name*

- Deadline: **Tomorrow – 16/09/2025**

Project

Discussion