Australian National University

Week 2 Announcements

- Wattle site "*Announcements*"… all good?
- We start labs this week! Lab #1 → participation marks.
  - **Only 10 minutes late for attendance!**
  - Exceptions: students with an approved EAP.
- Course representatives (candidates): 🙌 🤓
  - COMP2100: Sze Ming Chan.
  - COMP6442: Jiawei (Catherine) Ye, Yizhuo Gao.
  - Nominations close this **Friday, 4 August**, at noon.
- This Friday, 4 August, we'll have an online lecture!
- Education Access Plan (EAP) | Accessibility.

Accessibility

(formerly Access and Inclusion)

supports students at the Australian National University whose participation in academic studies is impacted by...

- Disability (physical or learning)
- Mental Health Condition/s
- Chronic medical condition/s
- Short-term illness/injury

Accessibility also supports:

- Carers
- Elite Athletes

If your circumstances are identified here and you require support to achieve your academic goals, please visit the Accessibility Website to find out about registering.

💻: website link here https://www.anu.edu.au/students/contacts/accessibility

Special Exam Arrangements (SEAs)

Deadline to renew EAPs and request SEAs for:

Semester 2 2023:

Mid-semester in-class/online assessments:
Friday, 4th August

End-of-semester exams:
Wednesday, 11th October

Students who do not inform Accessibility within the specified timeframes should be aware that their SEAs may not be implemented for the assessment or examination period.

New registrations after deadlines are considered on a case-by-case basis.

It is the student's responsibility to ensure that you have a valid Education Access Plan (EAP) in place with A&I at least two (2) weeks prior to examination periods if you want Special Exam Arrangements (SEAs), and that you contact your Course Convenor to arrange your SEAs.

Contact Us

Accessibility
University Experience Division
Di Riddell Student Centre (Level 3)
Acton, ACT 2601

Contact number: +61 2 6125 5036
Email link here: access.inclusion@anu.edu.au
Website link here: www.anu.edu.au/students/contacts/access-inclusion

Australian National University

Australian National University

HEX International: Singapore | Applications are open for the summer

- HEX Singapore is opening applications for its second offering of the year! Twenty ANU students will have the opportunity to start their entrepreneurial journey and get advice from the best entrepreneurs and industry stakeholders at HEX Singapore.
- This is a fantastic opportunity for students to take part in immersive real-world experiences in Lion City.
- Please reach out to comp.engagement@anu.edu.au for more information.
- HEX Singapore at ANU.

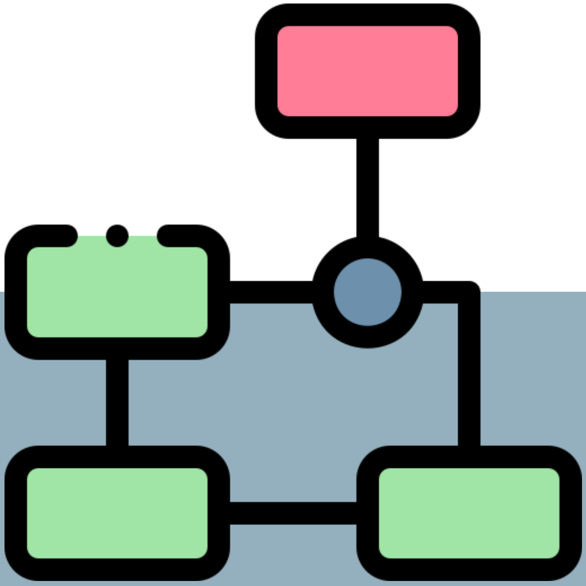
HEX
Hex Singapore
Fully funded places available for ANU Students
Twenty ANU Students will have the opportunity to start their entrepeneurial journey and get advice from the best entrepeneurs and industry stakeholders at HEX Singapore!
Applications close August 31

Australian National University

COMP2100/6442
Software Design Methodologies / Software Construction

UML

Bernardo Pereira Nunes and
Sergio J. Rodríguez Méndez



Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system. [3]

What is UML?

- Language for modelling systems
- Easy (“natural”) to understand and use
- Used for visualizing, specifying, constructing and documenting artifacts of software
- It allows understanding of a system from various perspectives
- UML is also used for documenting (e.g., requirements, design, architecture, …)
- UML **does not** provide algorithmic details

Sometimes used before coding (**forward design**)

Sometimes used after coding (documentation purposes) (**backward design**) (read reference [4])

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson. 2005. Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). Addison-Wesley Professional.
[4] Ana M. Fernández-Sáez, Michel R. V. Chaudron, Marcela Genero, and Isabel Ramos. 2013. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance? a controlled experiment. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13). ACM, New York, NY, USA, 60–71.

Australian National University

Unified Modeling Language (UML)

UML – Building Blocks

- Things (abstraction/elements that will be modelled)
- Relationships (relate things/tie them together)
- Diagrams (graphical representation of a set of elements (things and relationships))

> Diagrams:

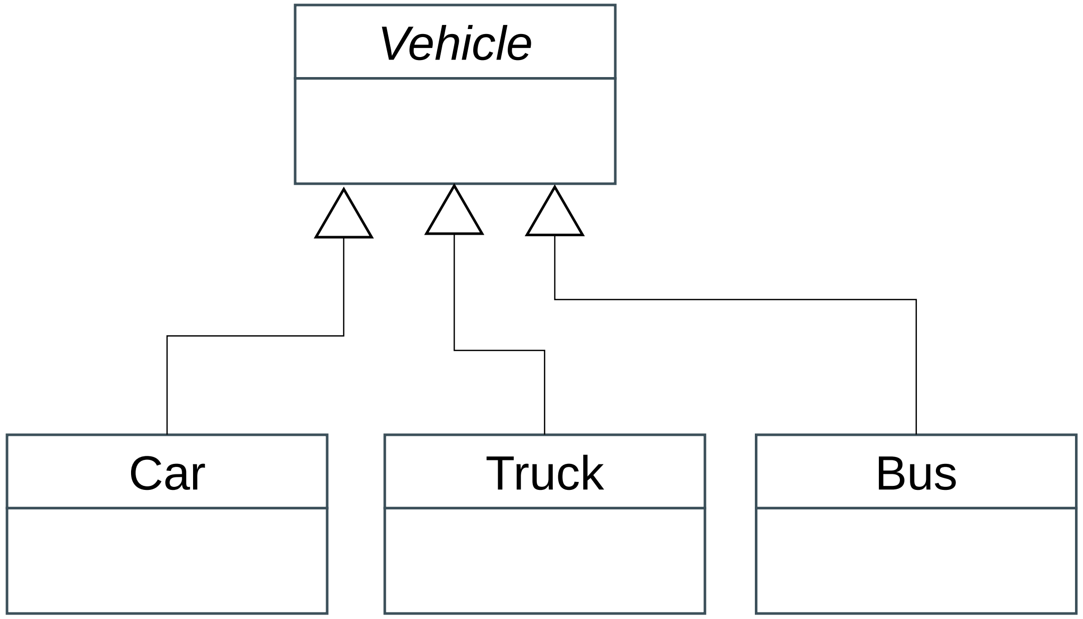
- Class Diagram
- Sequence Diagram
- Use Case Diagram
- … many others!

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson. 2005. Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). Addison-Wesley Professional.

Class Diagram

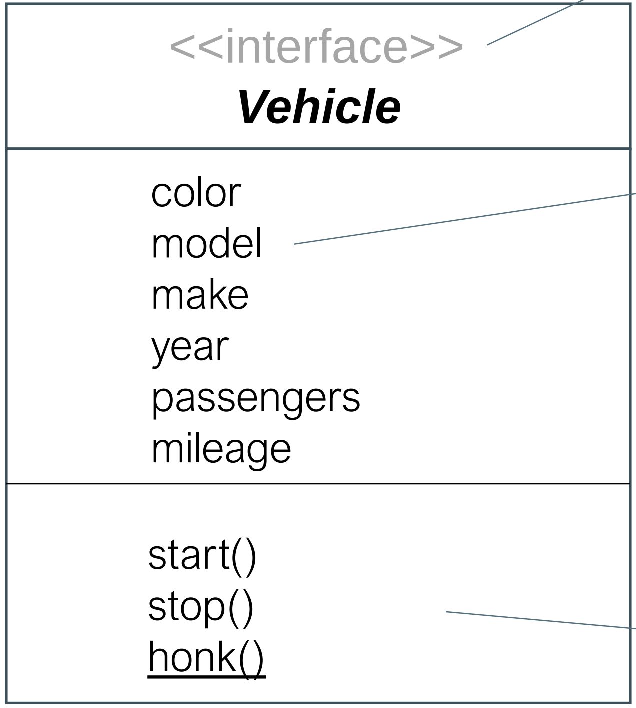
Object-Oriented Approach – Abstraction / Real-World Entities

“A class diagram shows a set of classes, interfaces, and collaborations and their relationships.”[3]



Extra reference: https://www.uml-diagrams.org/class-diagrams-overview.html

Class Diagram



Name (bold, capitalized, centered)
If it is an interface, add <<interface>>
if it is an abstract class, use italics (e.g., ***Vehicle***)

Attributes (not bold, lowercase, left-aligned)
It is optional.
[visibility] attributeName [[multiplicity]] [:type] [=initial value][{property}]

Examples:
+ year : Integer
- make : String {read only (final)}
- passengers : Person [0..N]
+ mileage : Double = 0.0 ← initial value

Operations
[visibility] operationName [(parameter-list)] [:return type] [{property}]
Use underlined for static methods (and attributes)

Examples:
+ start() : Boolean
- honk(level : Int) : void

*It presents some relevant items only.

Class Diagram

Visibility

	Modifier	Class	Package	Subclass	World
+	public	yes	yes	yes	yes
#	protected	yes	yes	yes	no
~	package	yes	yes	no	no
-	private	yes	no	no	no

*if no modifier is set, the default is **package**

Relationships

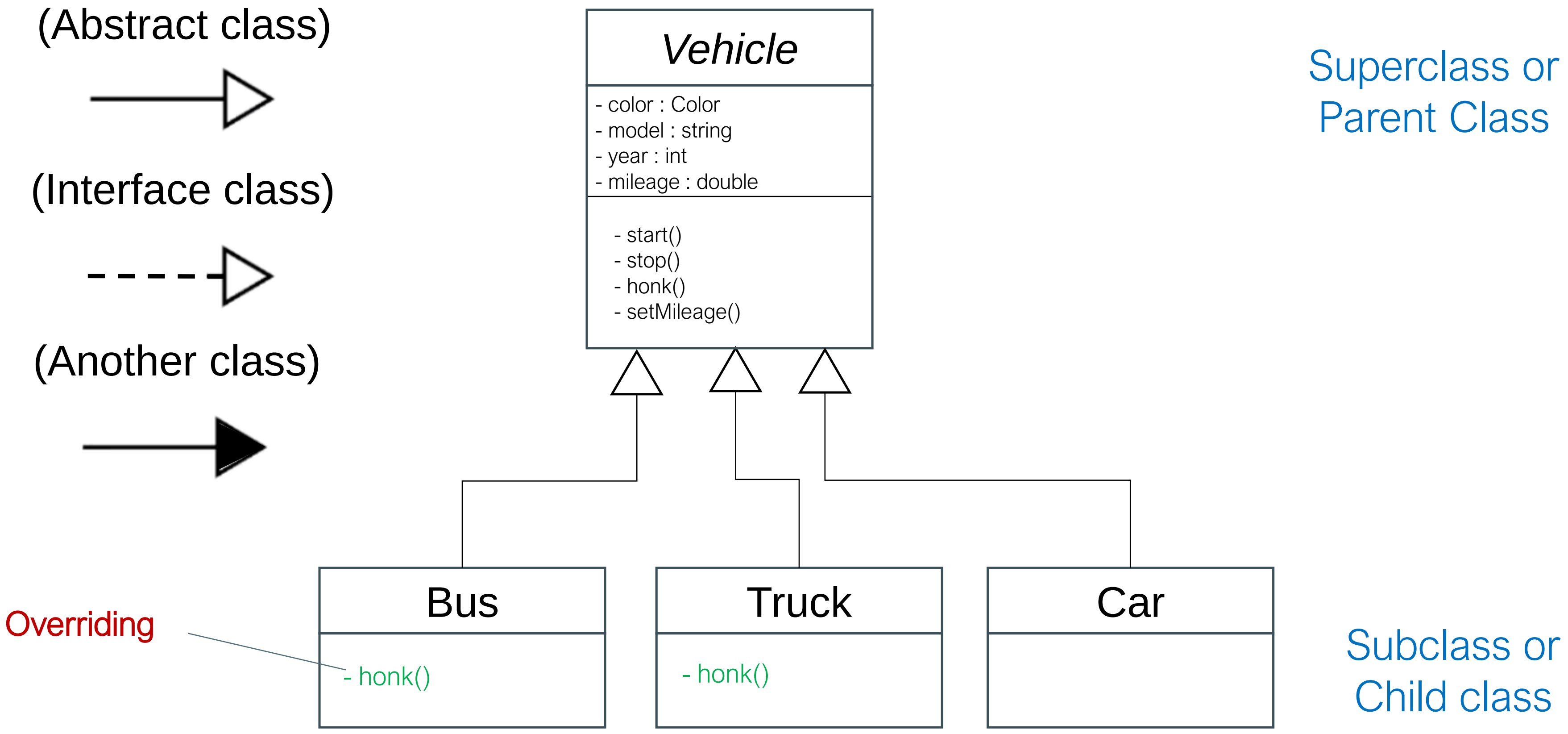
Generalizations (inheritance) – indicates a relationship between a more generalized class to more-specialized classes (**is-a; is-like-a**). The specialized class inherits all attributes, operations, and relationships defined in the more generalized class (parent-child relationship).

Associations – indicates structural relationships between instances. For example, an **employee works for an organization**. Each end of the relationship has properties (e.g., multiplicity).

- **Dependencies** – indicates runtime relation between classes (an object affects another object); one depends on another; a class uses objects (parameters) of another class (**uses-a**).
- **Aggregation** – indicates **is-part-of** / **has-a** relationships; object can exist outside the whole.
- **Composition** – indicates **whole-part** relations (**is-entirely-made-of**); the parts live and die with the whole (stronger form of aggregation)

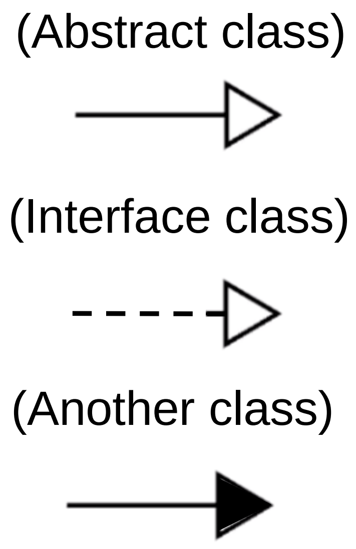
Relationships: Inheritance

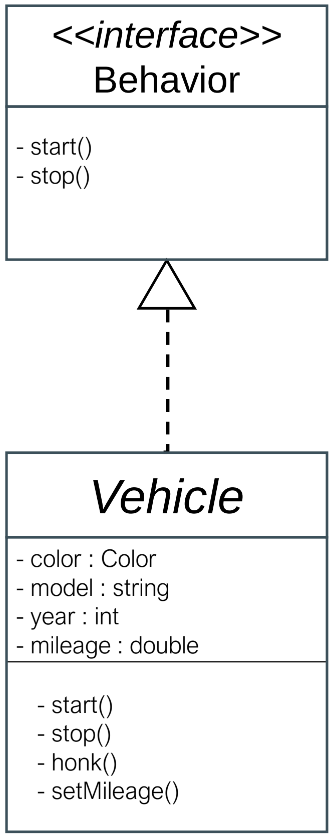
Generalizations (inheritance) – indicates a relationship between a more generalized class to more-specialized classes (is-a; is-like-a). The specialized class inherits all attributes, operations, and relationships defined in the more generalized class (parent-child relationship).

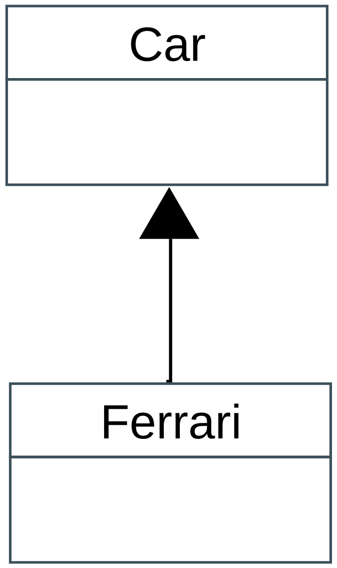


Relationships: Inheritance

Generalizations (inheritance) – indicates a relationship between a more generalized class to more-specialized classes (is-a; is-like-a). The specialized class inherits all attributes, operations, and relationships defined in the more generalized class (parent-child relationship).





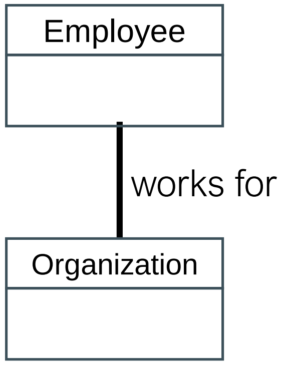


Relationships

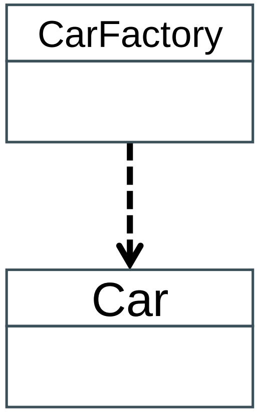
Associations – indicates structural relationships between instances. For example, an employee works for an organization. Each end of the relationship has properties (e.g., multiplicity).

- **Dependencies** – indicates runtime relation between classes; one depends on another; some class uses objects (parameters) of another class (**uses-a/depends-on/instantiate**).
- **Aggregation** – indicates **is-part-of** relationships; object can exist outside the whole.
- **Composition** – indicates **whole-part** relations (**is-entirely-made-of**); the parts live and die with the whole.

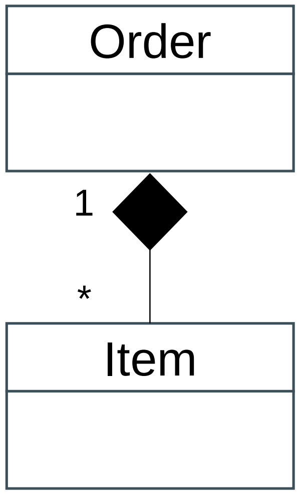
Association: Employee works for Organization



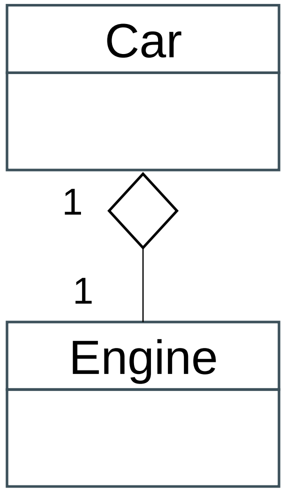
Dependency: CarFactory depends-on Car



Composition: Order is-entirely-made-of Items



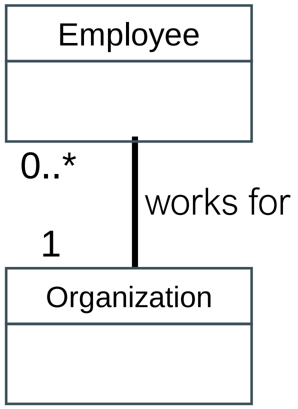
Aggregation: Engine is-part-of Car

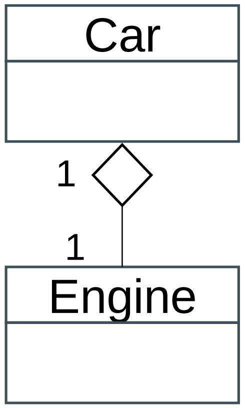


Multiplicity

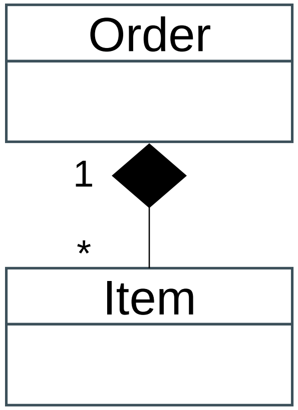
Associational relationships

Multiplicity	Meaning
0..1	zero or one instance. The notation n.. m indicates n to m instances.
0..* or *	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance



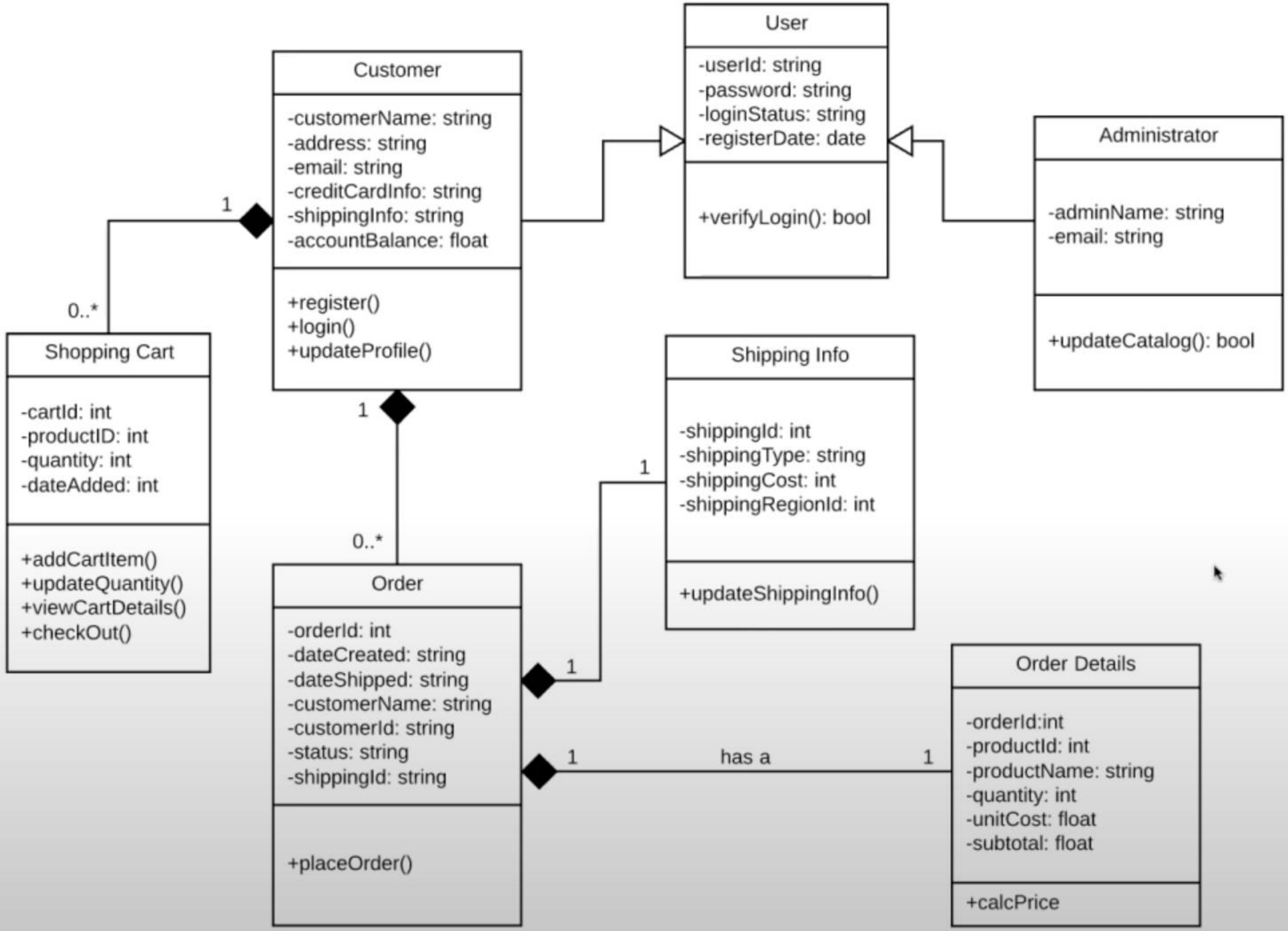


one-to-one



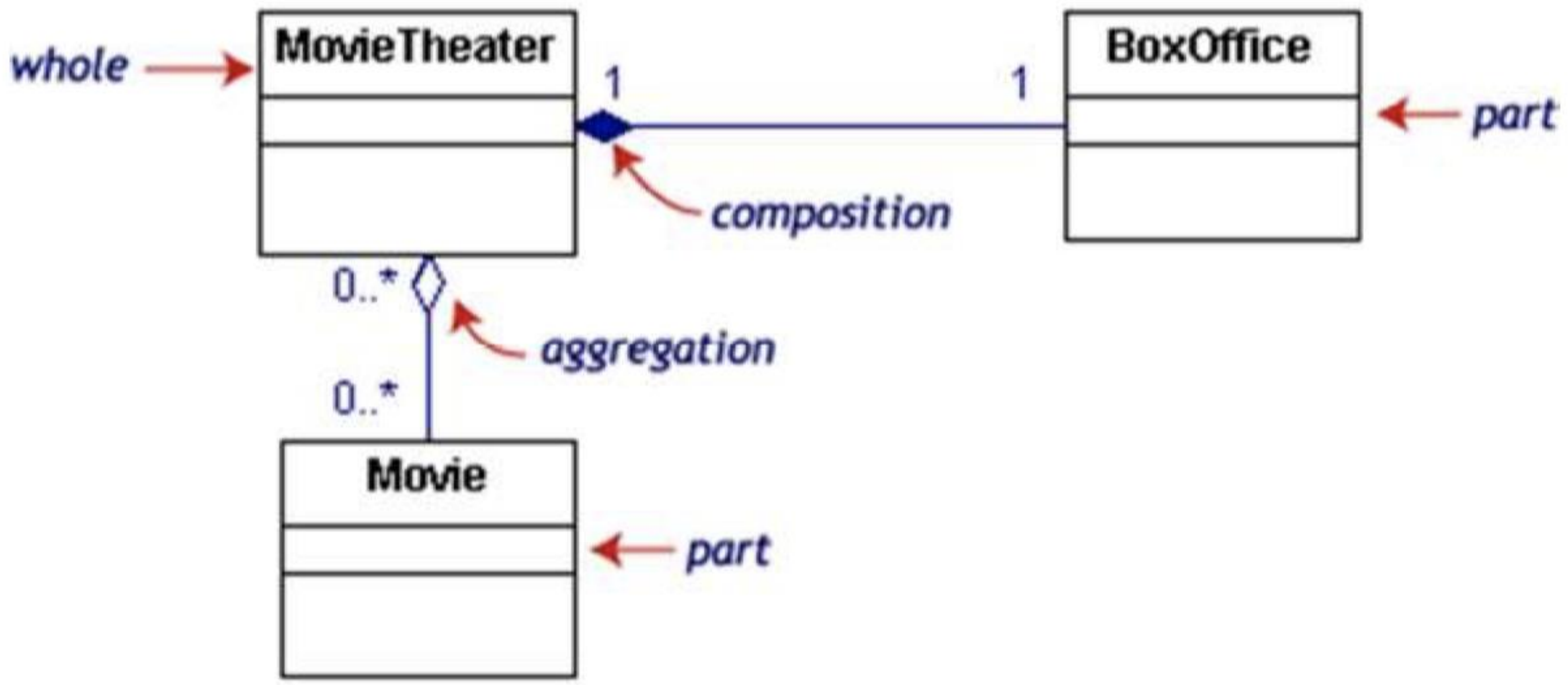
one-to-many

Source: http://edn.embarcadero.com/article/31863



source: https://www.lucidchart.com

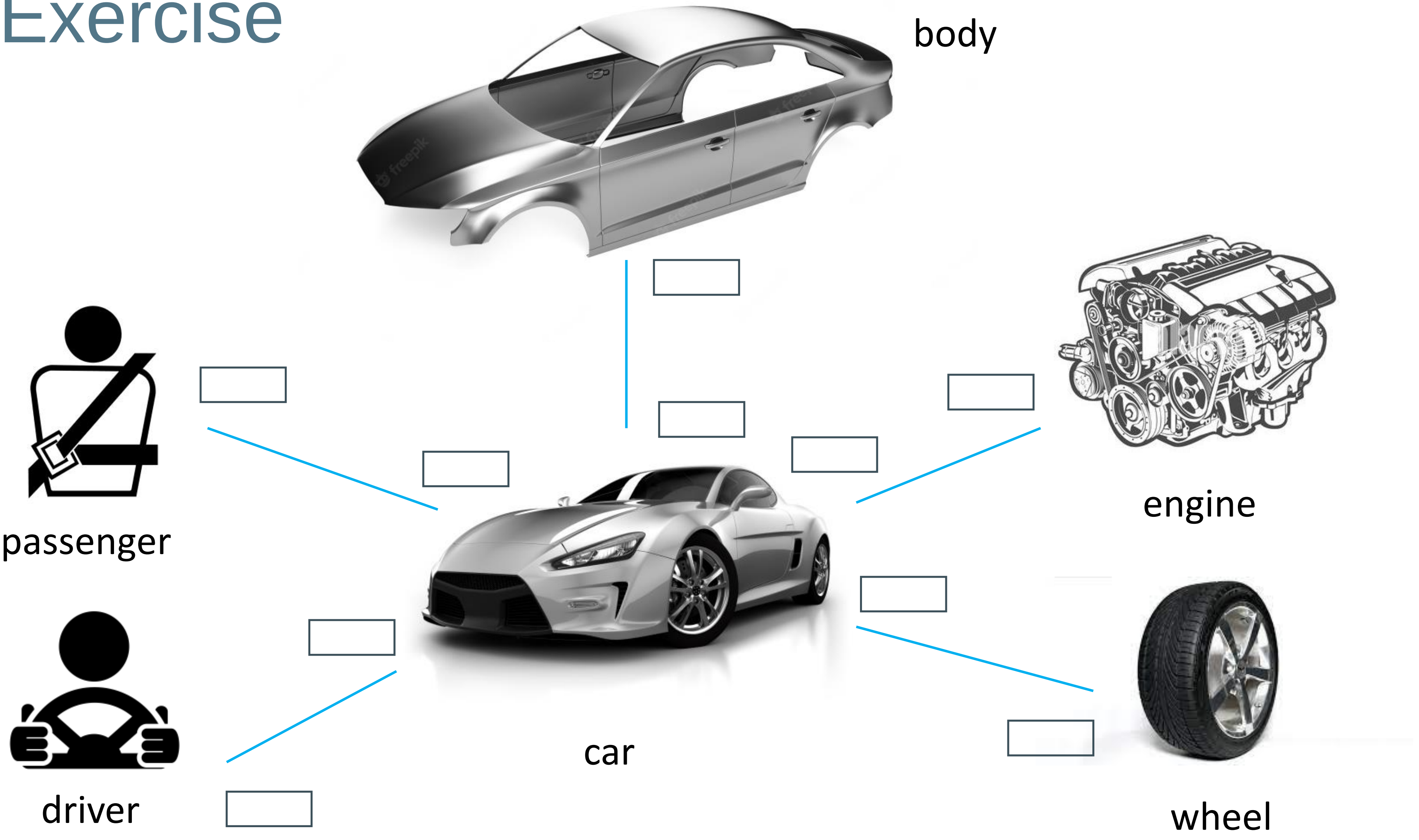
Example



MovieTheater is destroyed, BoxOffice is also destroyed

Movie still exists

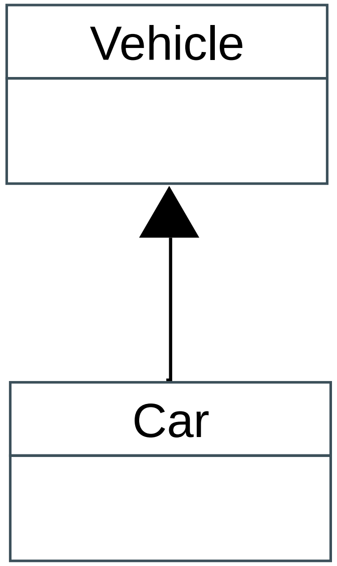
Exercise



Java Inheritance

- Subclass inherits features (attributes and operations) from Superclass
- Reusability (reuse instead of rewriting operations and attributes in a subclass)
- Subclass can add new attributes and methods (extend)
- “IS-A” or Parent-Child relationship

```
class Vehicle {
		//TODO
}


class Car extends Vehicle{
		//TODO
}
```



Java Inheritance

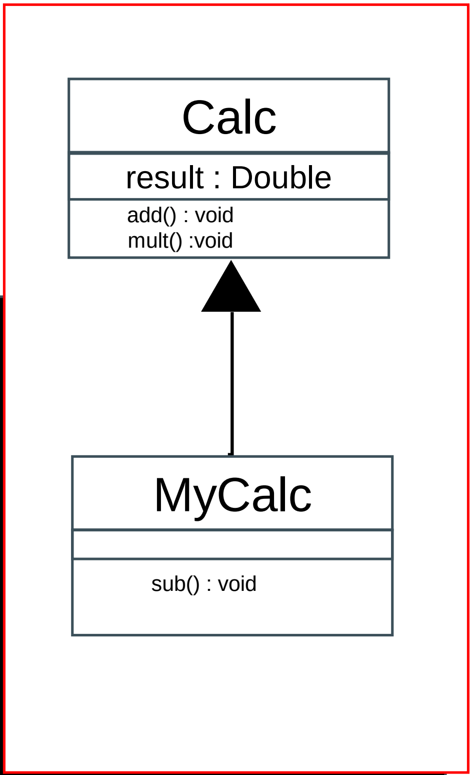
- A common example using inheritance:

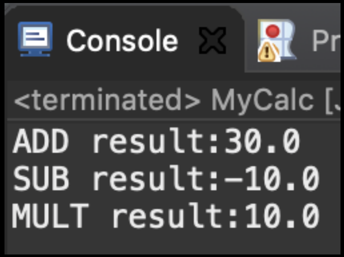
```
class Calc { //superclass
  double result;

  public void add(double x, double y) {
    result = x + y;
    System.out.println("ADD result:" + result);
  }

  public void mult(double x, double y) {
    result = x * y;
    System.out.println("MULT result:" + result);
  }
}
```

Java Inheritance

- A common example using inheritance:

```
public class MyCalc extends Calc {//subclass
  public void sub(int x, int y) {
    result = x - y;
    System.out.println(”SUB result:" + result);
  }


  /* driver */
  public static void main(String args[]) {

    MyCalc mc = new MyCalc();
    mc.add(10, 20); //mc inherits add
    mc.sub(10,20);
    mc.mult(2, 5); //mc inherits mult
  }
}
```



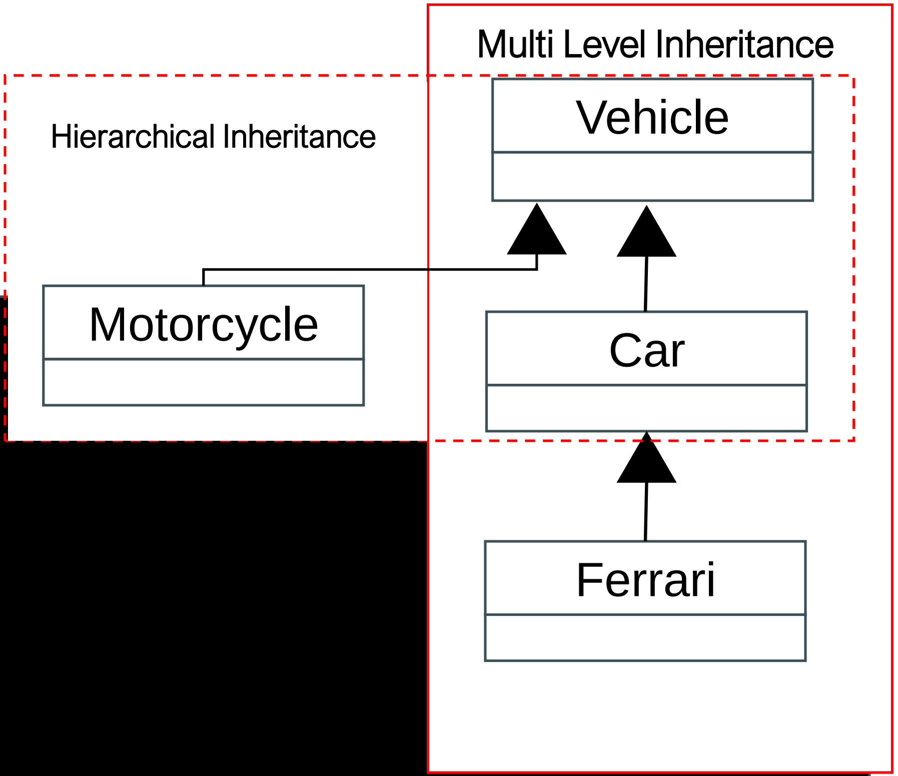


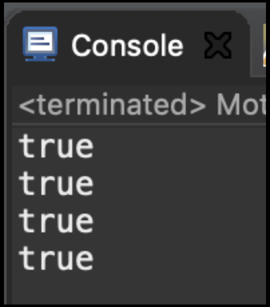
```
<terminated> MyCalc [
ADD result:30.0
SUB result:-10.0
MULT result:10.0
```

Java Inheritance

- Another example using inheritance:

```
class Vehicle {
}

class Car extends Vehicle {
}

class Ferrari extends Car {
}

public class Motorcycle extends Vehicle {

/* driver */
public static void main(String args[]) {
    Vehicle    v = new Vehicle();
    Car        c = new Car();
    Ferrari    f = new Ferrari();
    Motorcycle m = new Motorcycle();

    System.out.println(v instanceof Vehicle);
    System.out.println(c instanceof Car);
    System.out.println(f instanceof Ferrari);
    System.out.println(m instanceof Motorcycle);
    //m instanceof Vehicle?
  }
}
```

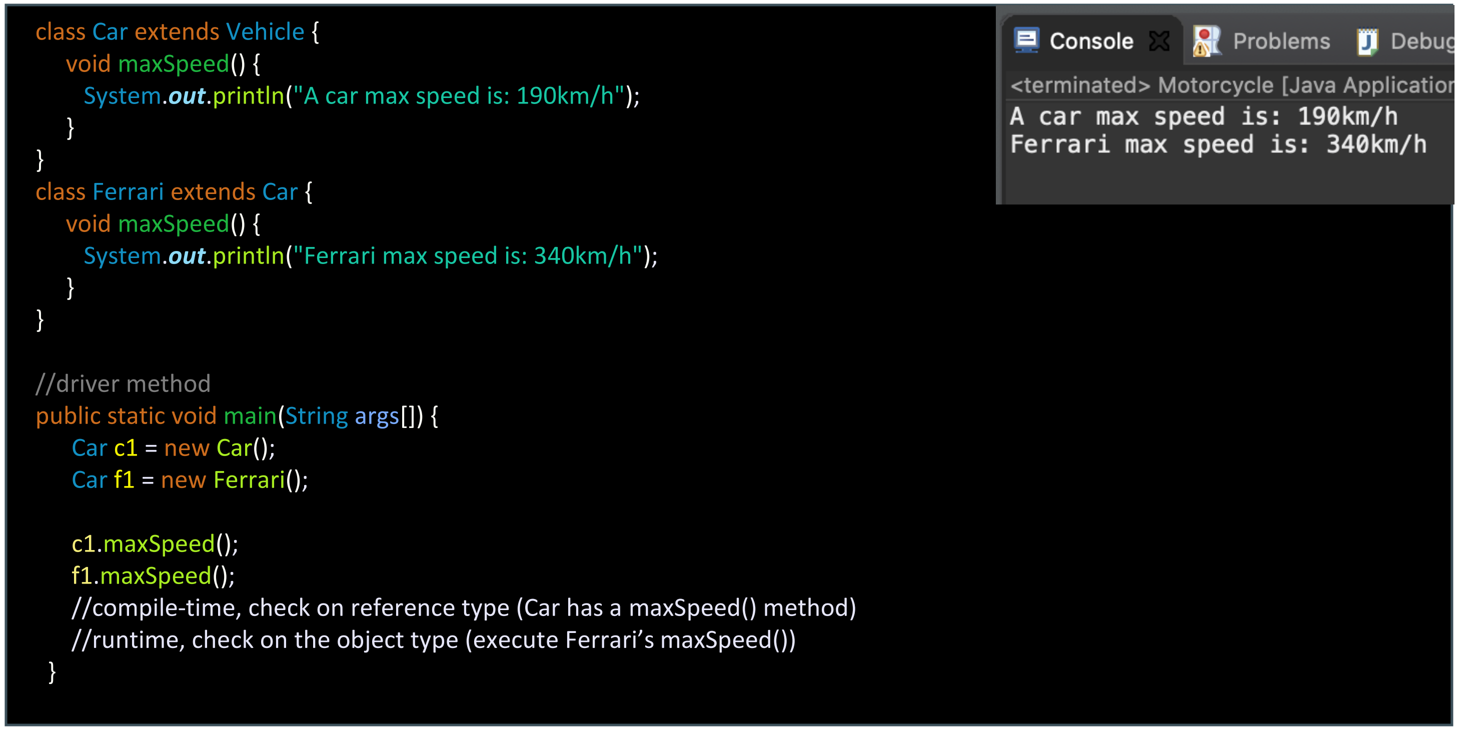




```
Console
<terminated> Mot
true
true
true
true
```

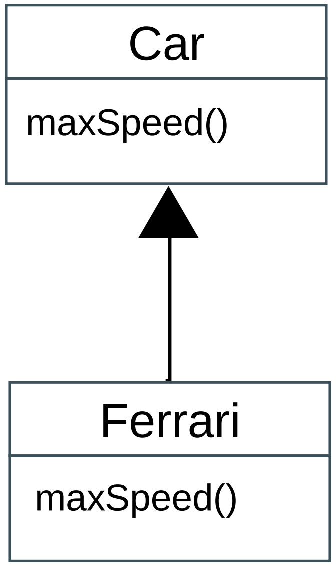
Java Overriding

- Inheritance allows the subclass to override an existing superclass method (*unless it is **final***)
- It redefines superclass behavior to meet subclass requirements



```java
class Car extends Vehicle {
    void maxSpeed() {
      System.out.println("A car max speed is: 190km/h");
    }
}
class Ferrari extends Car {
    void maxSpeed() {
      System.out.println("Ferrari max speed is: 340km/h");
    }
}

//driver method
public static void main(String args[]) {
     Car c1 = new Car();
     Car f1 = new Ferrari();

     c1.maxSpeed();
     f1.maxSpeed();
     //compile-time, check on reference type (Car has a maxSpeed() method)
     //runtime, check on the object type (execute Ferrari's maxSpeed())
  }
```

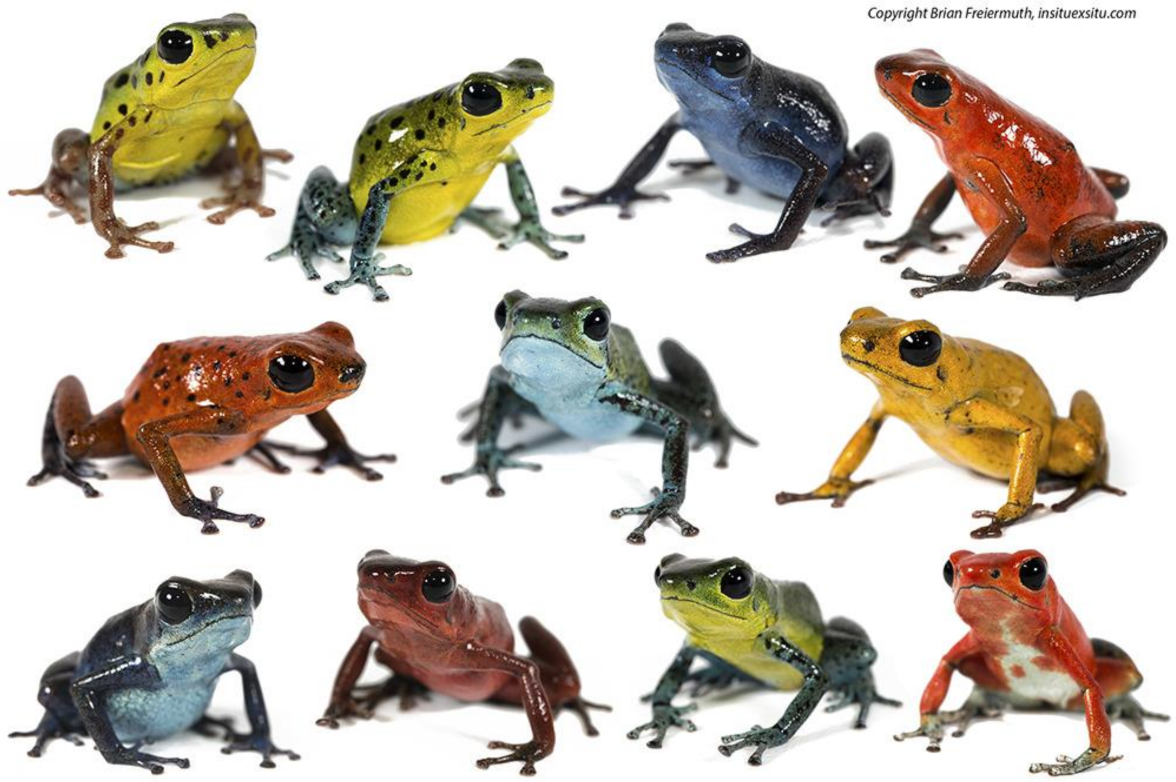
```
<terminated> Motorcycle [Java Application
A car max speed is: 190km/h
Ferrari max speed is: 340km/h
```



Australian National University

Polymorphism

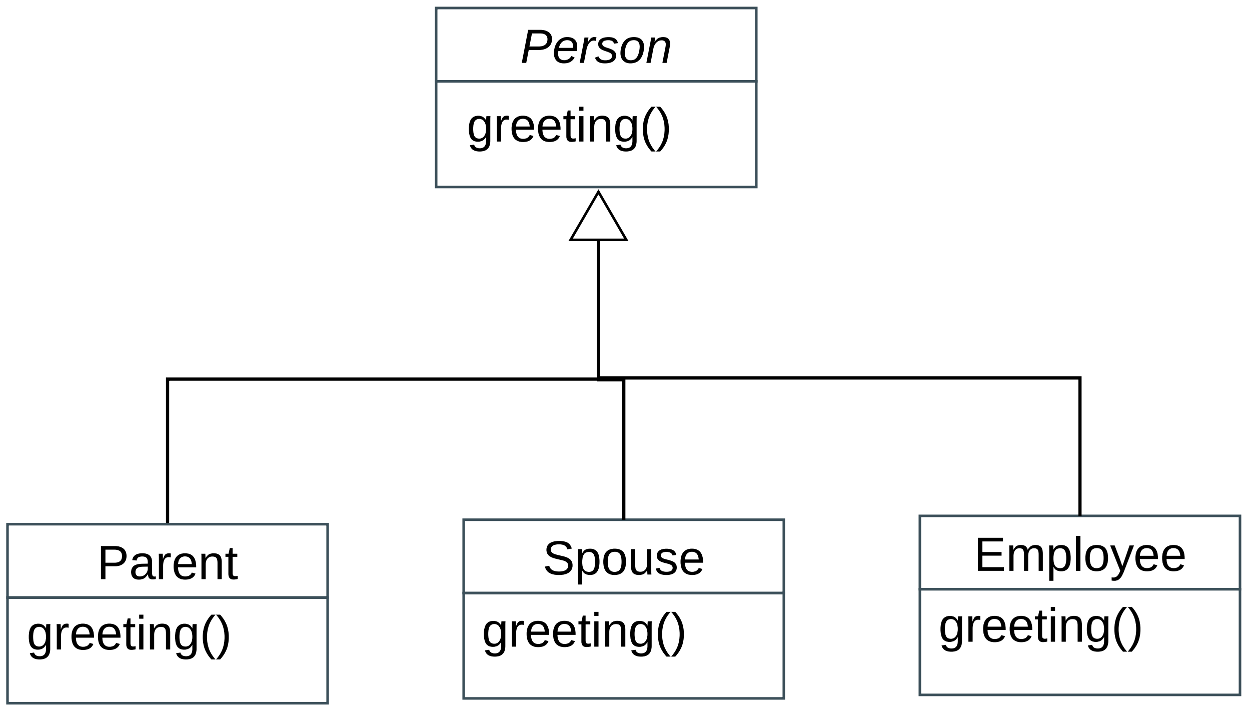
"Polymorphism refers to a principle in biology in which an organism or species can have many different forms or stages." **poly** -> many / **morphs-**> forms



All frogs are of the **same specie** – *strawberry poison dart frog* (color polymorphism).

Polymorphism

Different behaviors in different situations

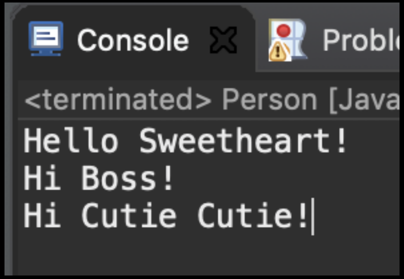


(*UML does not represent polymorphism. The diagram represents inheritance only).

Polymorphism

```
import java.util.ArrayList;

public abstract class Person {

  void greeting() {}

public static void main(String args[]) {

  ArrayList<Person> ap = new ArrayList<Person>();
  ap.add(new Spouse()); //upcasting
  ap.add(new Employee()); //upcasting
  ap.add(new Parent()); //upcasting

  for (Person p : ap)
    p.greeting();
 }
}
```

```
class Spouse extends Person{
  void greeting(){
    System.out.println("Hello Sweetheart!");
  }
}

class Employee extends Person{
  void greeting(){
    System.out.println("Hi Boss!");
  }
}

class Parent extends Person{
  void greeting(){
     System.out.println("Hi Cutie Cutie!");
  }
}
```



```
<terminated> Person [Java
Hello Sweetheart!
Hi Boss!
Hi Cutie Cutie!
```

Inheritance vs Polymorphism

Some differences that may help to understand both concepts:

Inheritance

- Reusability (inherit and reuse parent class features [operations and attributes])
- Classes (new classes are created based on the parent class)

Polymorphism

- Object decides which form to take (at compile-time or at runtime)
- Applied to operations (can take different behaviors/forms)

Exercise

Which alternative is **INCORRECT** about software construction:

Select one:

1. Reuse often involves the use of libraries. It enables higher productivity, and probably cost reductions in software construction.

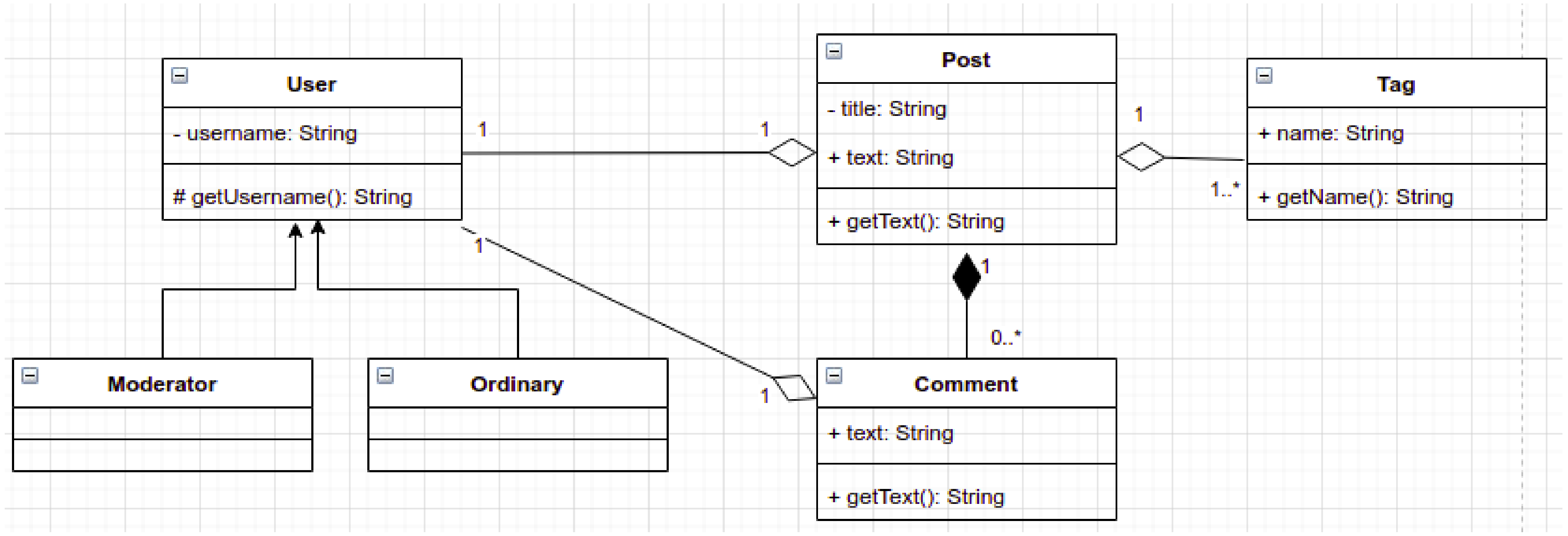
2. Integrated Development Environments (IDEs) help developers during software construction. IDEs usually come with features such as code editor, debugging tool, and source code integration.

3. Writing classes, naming variables and creating control structures are examples of tasks related to software construction.

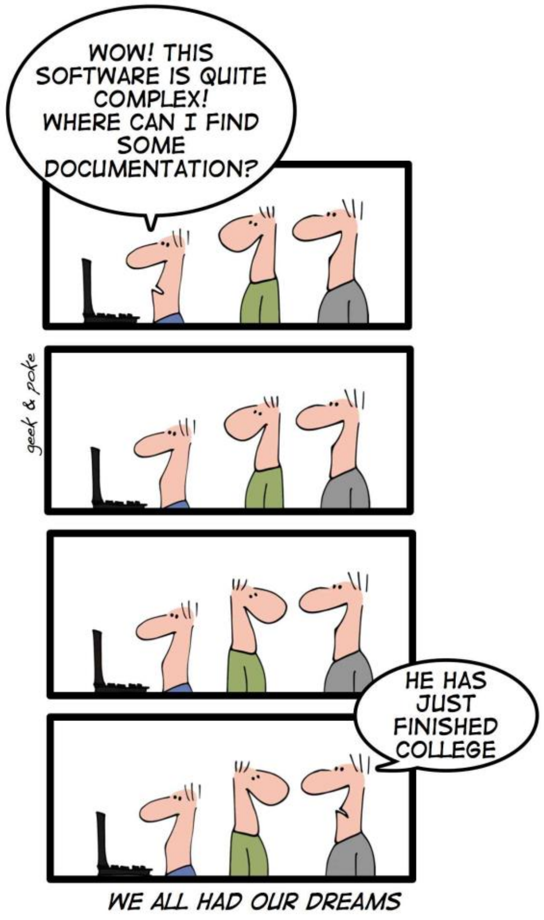
4. Whenever you have extensive software documentation, code readability is less important in software construction. This is because good documentation is enough to guide developers in creating and maintaining applications.

Exercise

Considering the UML class diagram below, explain:
i) the different types of relationships
ii) the multiplicity between the classes
iii) the visibility of the methods and attributes



Meme for today's lecture! Keep practicing!


WOW! THIS SOFTWARE IS QUITE COMPLEX! WHERE CAN I FIND SOME DOCUMENTATION?
geek & poke
HE HAS JUST FINISHED COLLEGE
WE ALL HAD OUR DREAMS

Main References

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson. 2005. Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). Addison-Wesley Professional.

[2] UML Specification: https://www.omg.org/spec/UML/2.5.1/PDF