Object Oriented Analysis & Design

Lesson 1 - Week 1

Introduction to Object Orientation

Lecturer: Dr. Msagha J Mbogholi, PhD

Content

- Introduction
- Tracing the Object Oriented Paradigm
- Foundations of Object Orientation
- Object Oriented Concepts
- Strengths and weaknesses
- Application areas

Part 1

Introduction

Introduction

- As of April 2022, 63% of the global population was using the Internet (Statista Research Department, 2022). This goes to show how we have become so dependent on the Internet and different applications to solve our day to day problems.
- This ranges from the use of Google (now an accepted English word meaning 'to search for' something, to cloud computing solutions, to Internet of Things (IoT) devices, and offline corporate and retail programs (solutions).
- Have you ever thought about how much work is put in to develop that application that you find so convenient to solve your day to day problems? For example applications that you download from the store for free?
- Thankfully as a user one does not need to know the nitty gritty of the development of these applications on their phones (or any other device for that matter). What interests the user is that it works, and is easy to use, right?

Introduction (cont'd)

- The general term used to refer to the different programs that you use whether off or online is software. These are the engines that run the applications and games that you use on a day to day basis.
- With time programming languages have evolved to the point that programmers can write very complex solutions while enabling you (the user) to only see the details that you need to see to solve the particular problem. The use of graphical user interfaces has become such a crucial factor that there is a whole domain called user experiences (UX) that is exclusively dedicated to that.
- However, every program is written to offer a solution to an identified problem. There is a process that is followed in arriving at this solution which is covered in greater detail in a course called software engineering.
- This course, nonetheless, is concerned with an approach called the object oriented approach. In this course the learner will understand how the object oriented approach is used and why it is the approach of choice for today's software engineer (as mentioned earlier there are other approaches as well).
- Every lesson is designed to build up on the previous one to enable the learner follow the new concepts introduced and also understand the different tools and concepts in object oriented analysis and design.

Part 2

The Object Oriented Paradigm

2.1 Introduction

- That one is able to confidently use software that has been designed to be so easy to use attests to the strength of object oriented programming, which in itself is a consequence of good object oriented analysis and design. The relationship between the two shall be discussed in detail in later lessons.
- In this lesson a trace to the object oriented (OO) paradigm is done, as well as an introduction to foundation, concepts and technologies associated with OO. A discussion of strengths and weaknesses of OO is also included.

2.2 Tracing the OO Paradigm

- A paradigm can be simply explained as a systematic way of doing things; a model or pattern in other words.
- In designing solutions designers have adopted to certain paradigms over time; these paradigms are implemented using the different programming languages that have evolved progressively with them.
- Fig 1 describes the general evolution of the paradigms over time. Four eras (paradigms) are identified: procedural paradigm, modular paradigm, abstract paradigm and the object oriented paradigm (Ojo and Estevez, 2005).

2.2 Tracing the OO Paradigm (cont'd)

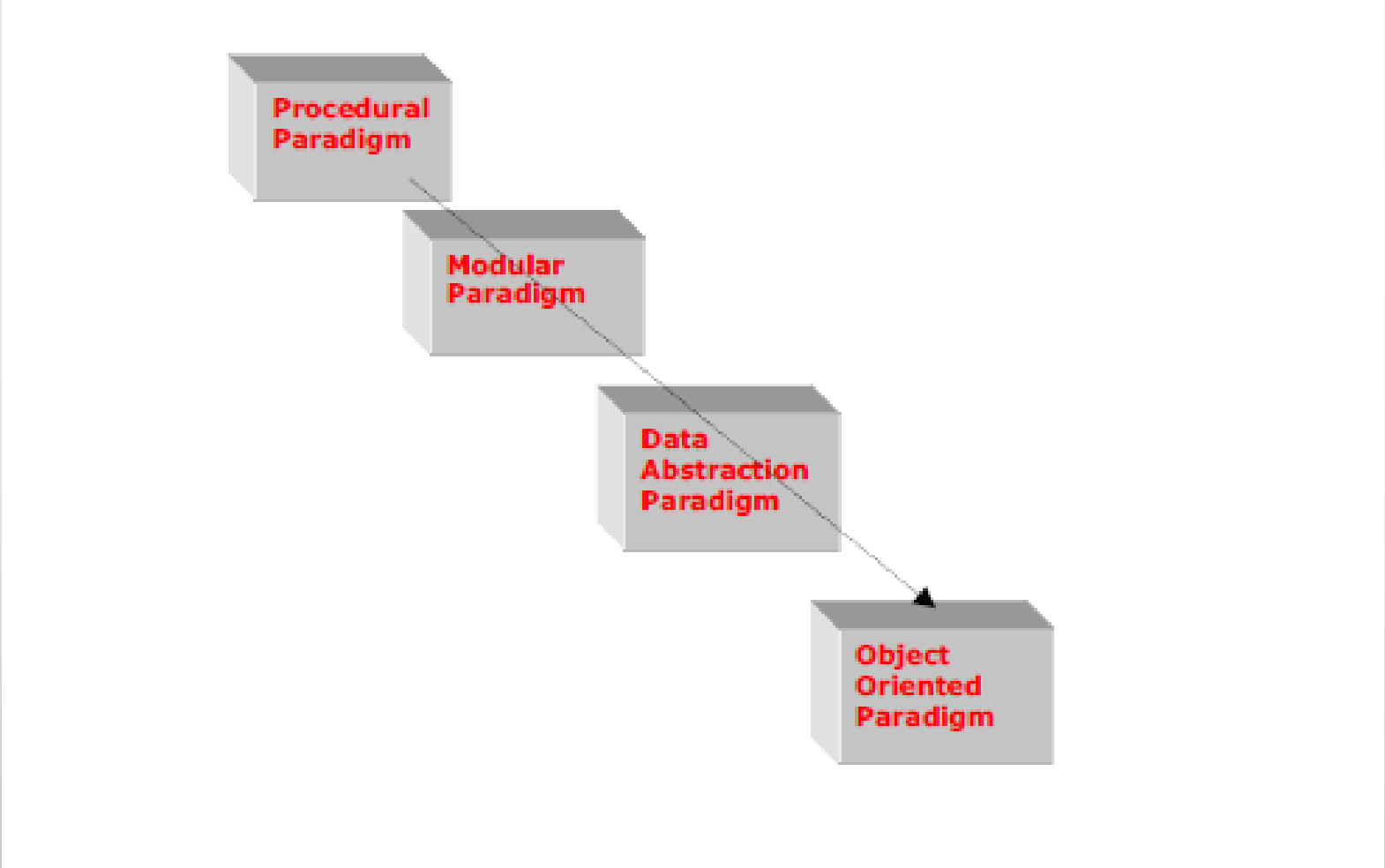

Fig 1. Tracing the OO Paradigm (Ojo & Esteveez, 2005)

2.2 Tracing the OO Paradigm

- **Procedural paradigm:** this is an old paradigm based on the early ways of writing programs. Just as the name implies, procedural programs were based on a step by step approach to arriving at solutions. Thus the program would follow a "procedure", in this case a series of statements executed sequentially, to arrive at the solution. The procedure would have to be called in order for it to execute. The programs also just worked with the state of the machine they were resident in, and were written to solve specific problems. The advantage to this approach is that it is quite simple; conversely, it can't be used to solve complex problems, and it is less efficient compared to later paradigms. Examples of languages applying this paradigm include BASIC, Fortan, C and COBOL.

2.2 Tracing the OO Paradigm

- **Modular Paradigm:** this paradigm is also known as the **structured programming** paradigm. One of the weaknesses of imperative programming was the use of a command called "goto" which was the only way program execution could be forced to "jump" from a line of code to another line which was not the next one. As program complexity increased it became impractical to use this statement. Arguments were made by different programmers notably Bachmann and Dijkstra (1966) who argued for an alternative to this approach. This introduced modular (structured) programming to solve this problem. The approach is also referred to as a top down approach as solutions are developed from a general then specific, solutions. The approach was to bring "structure" to a program using three constructs:
- ***Sequence:*** this is where a series of commands are executed in sequence (one line at a time), for example consider the steps taken to go to school:
- Wake up, take a shower, brush teeth, eat breakfast, brush teeth (again), pick school bag, go to bus stage, board bus, arrive at school.
- These are actions that you perform sequentially, arriving at the final target which is to go to school.

2.2 Tracing the OO Paradigm

- ***Repetition:*** repeat an action (statement) while a certain condition remains true (or until it stops being true). A good example of this is when you go to your favorite game arcade, or when playing your favorite game at home from your PS or Xbox console. We can write it as follows:
- For arcade scenario:

Arrive at game arcade,

Buy tokens and put in pocket,

While pocket is not empty,

    Play games

End while

Leave arcade, board bus

Go home

* Can you try and rewrite this for the "playing your favorite game at home" scenario?

2.2 Tracing the OO Paradigm

- ***Selection:*** select (choose) just one action from a set or list of choices. For example you are feeling hungry and want to spoil yourself during lunch time.
- Go to ATM
- Check account balance
- If balance less than 100
  - Eat fries only
- Otherwise if more than 100 but less than 200
  - Eat fries and burger
- Otherwise if more than 200
  - Eat fries, burger and ice cream

2.2 Tracing the OO Paradigm (cont'd)

- Further, modular programming also allows the use of modules; thus it means that the program can be broken down into several modules. Each module can then fulfill a particular function, making it easier to break down the requirements and fulfill them using different modules (the modules together fulfill all the requirements).
- Having the code (functionality) well structured (modular) makes it easier to read (and debug too), test, makes it easy to find code and allows for reusability.

2.2 Tracing the OO Paradigm

- **Data Abstraction Paradigm:** This paradigm opened the door to the object oriented paradigm and is indeed in use therein. The principle of abstraction is also in use in many other areas notably in database management structures. The concept is to hide details of the implementation at various levels. For example in database systems there is a 3 layer hierarchy of abstraction namely, physical, conceptual, and view levels. What this means is that at every layer there is either increasing abstraction (less details) or decreasing abstraction (more details).
- The reasoning behind abstraction is to hide details from the user which they do not need to know. This will be described in greater detail later on in this lesson.

2.2 Tracing the OO Paradigm

- **Object Oriented Paradigm:** this paradigm is based on real world entities which are referred to as objects. The focus using this approach is to solve real world problems by focusing on the characteristics and actions (behaviors) that these objects perform. The OO paradigm is the basis of this course and is the most popular approach used to solve real world problems using software solutions. The main advantages of writing code using this approach is that they are maintainable, reusable and more scalable compared to other approaches. In a nutshell, systems built using this approach are built around the objects that will be represented in them. This approach utilizes the bottom up concept where the design starts with the smallest components and these together make up the larger solution; thus there are many sub-systems making up one large system. This will also be discussed later on in the lesson and the course.
- Table 1 compares solution development (programming) based on the structural and object oriented paradigms.

Structured Programming	Object Oriented Programming
Structured Programming is designed which focuses on process/ logical structure and then data required for that process.	Object Oriented Programming is designed which focuses on data.
Structured programming follows **top-down approach**.	Object oriented programming follows **bottom-up approach**.
Structured Programming is also known as **Modular Programming** and a subset of procedural programming language.	Object Oriented Programming supports **inheritance, encapsulation, abstraction**, **polymorphism**, etc.
In Structured Programming, Programs are divided into small self contained functions.	In Object Oriented Programming, Programs are divided into small entities called **objects**.
Its main aim is to improve and increase quality, clarity, and development time of computer program.	Its main aim is to improve and increase both quality and productivity of system analysis and design.
It simply focuses on functions and processes that usually work on data.	It simply focuses on representing both structure and behavior of information system into tiny or small modules that generally combines data and process both.
It provides less flexibility and abstraction as compared to object-oriented programming.	It provides more flexibility and abstraction as compared to structured programming.
It is more difficult to modify structured program and reuse code as compared to object-oriented programs.	It is less difficult to modify object-oriented programs and reuse code as compared to structured programs.
It gives more importance of **code.**	It gives more importance to **data.**
Structured Programming is less secure as there is no way of data hiding.	Object Oriented Programming is **more secure** as having **data hiding** feature.
Structured Programming can solve moderately complex programs.	Object Oriented Programming can solve any **complex** programs.
Less abstraction and less flexibility.	More abstraction and **more flexibility**.

Table 1. Comparison of structured and object oriented programming (vedant-morkar19.medium.com)

Part 3

Foundations of Object Orientation

3.1 Introduction

- Ojo and Estevez (2005) describe object orientation as "viewing and modelling the world/system as a set of interacting and interrelated objects".
- They further emphasize that the OO approach is twofold: that, "
- the universe consists of interacting objects, and
- (it) describes and builds systems consisting of objects" (Ojo and Esteveez, 2005)
- Fig 2 shows the foundations on which the OO approach is built.

3.2 Foundations of Object Orientation

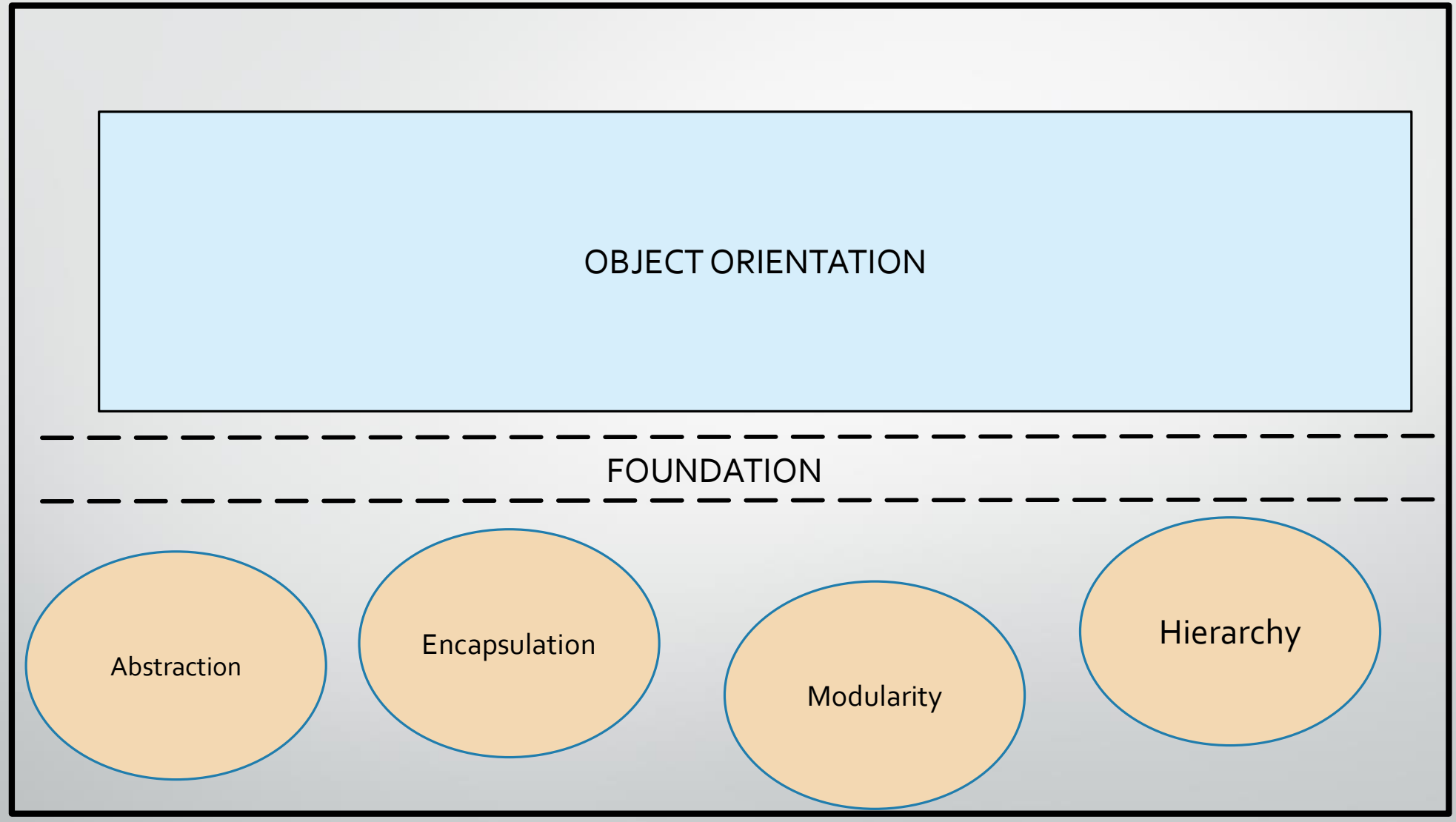

Fig 2. Foundations of object orientation

3.2.1 Abstraction

- This principle is based on what you can call “need to know”.
- It entails leaving out details that aren’t important to either:
  - Obtain a solution to the problem,
  - Differentiate one object from another.
- A good example is like when you go shopping. Do you know the details of every salesperson you meet? Their age, weight, shoe size, and the like? Of course not; perhaps the name only so that you can remember who served you (or to make them feel good too). The other details aren’t important in order for you to get assistance in purchasing.
- How about in school; what details do you need to know about your professor? I have students who still don’t know my name by the end of the semester and so have difficulty finding my office!

3.2.1 Abstraction

- Also think about the products you buy in the supermarkets you visit; how much do you need to know about the food or electronics you buy? Ever heard a customer ask what food a chicken was fed on in order to determine whether to buy it or not? Or what materials were used on the circuit board of the pocket radio you wish to buy?
- Abstraction therefore, helps to manage the complexity by doing away with details that aren't important in finding the solution to a problem and focusing on what differentiates objects.
- Without abstraction developing solutions using the OO approach would be cumbersome and tiring; not to mention the possibility of even developing the wrong solution due to too much detail (what is also known as information overload).

3.2.2 Encapsulation

- Ashrafi and Ashrafi (2014) defines encapsulation as "the packaging of data and processes within one single unit." Let us demonstrate this using a few real world examples.
- When I was younger I would take a public bus to school. In my country automatic transmission vehicles were introduced later than the rest of the world. I would marvel at how the driver would just focus on steering and breaking without having to change gears. If you have the same fascination as me (or you perhaps drive) have you thought about what is involved in just putting a car on "D" and then driving off? These details are lost on you since you just step on the gas and focus on steering. But the motions of that gearbox to changing gears and all the things you don't see are lost on you so that you can focus on more pertinent issues (like ensuring you don't drive into a ditch).
- Another good example is the TV remote. You press the volume up or down button as you desire and observe the change; what happens inside the remote is not of concern to you as long as it works. You can change the channels and perform any action you desire and the TV will respond accordingly.

3.2.2 Encapsulation

- Similar to encapsulation is information hiding. The principle is the same but implementation is slightly different. Information hiding is more concerned with protecting the data from careless and/or malicious interference (intentional or not). A good example of this is the ATM, where you can only interact through the designed interface. Your operations are restricted to what it will allow you to do; for example, you can't change your available balance even though you can see it. Your data is therefore protected.
- This concept allows us to introduce the term black box. Encapsulation and information enable an entity (real world object) to become a black box; this means that the space that entity lives in can be divided into the inside (which is private and can't be accessed directly without the object's permission) and public (which is accessible to all).
- A good example of this is your own body; the inner workings are private to everyone. However, what people interact with is what is visible to them. How your brain processes things for example is private to you; another person can only know what you're processing if you tell them so.

3.2.3 Modularity

- Modularity refers to breaking down a complex system into smaller units that are more manageable. These smaller units are usually built around functionality and are fully functional on their own; thus they can be managed independently.
- Consider an example of a car. The car can be broken down into modules such as engine and body modules. Engine modules can be further broken down into mechanical and electrical modules. Body parts can be broken down into passenger, boot and nose. Thus each module can be handled as a unit on its own. This makes it easier to manage and proper attention can be given to individual modules. The modules can then be integrated to make up the larger system.
- This can be demonstrated using fig 3; it shows the breakdown of the car into different modules.

3.2.3 Modularity

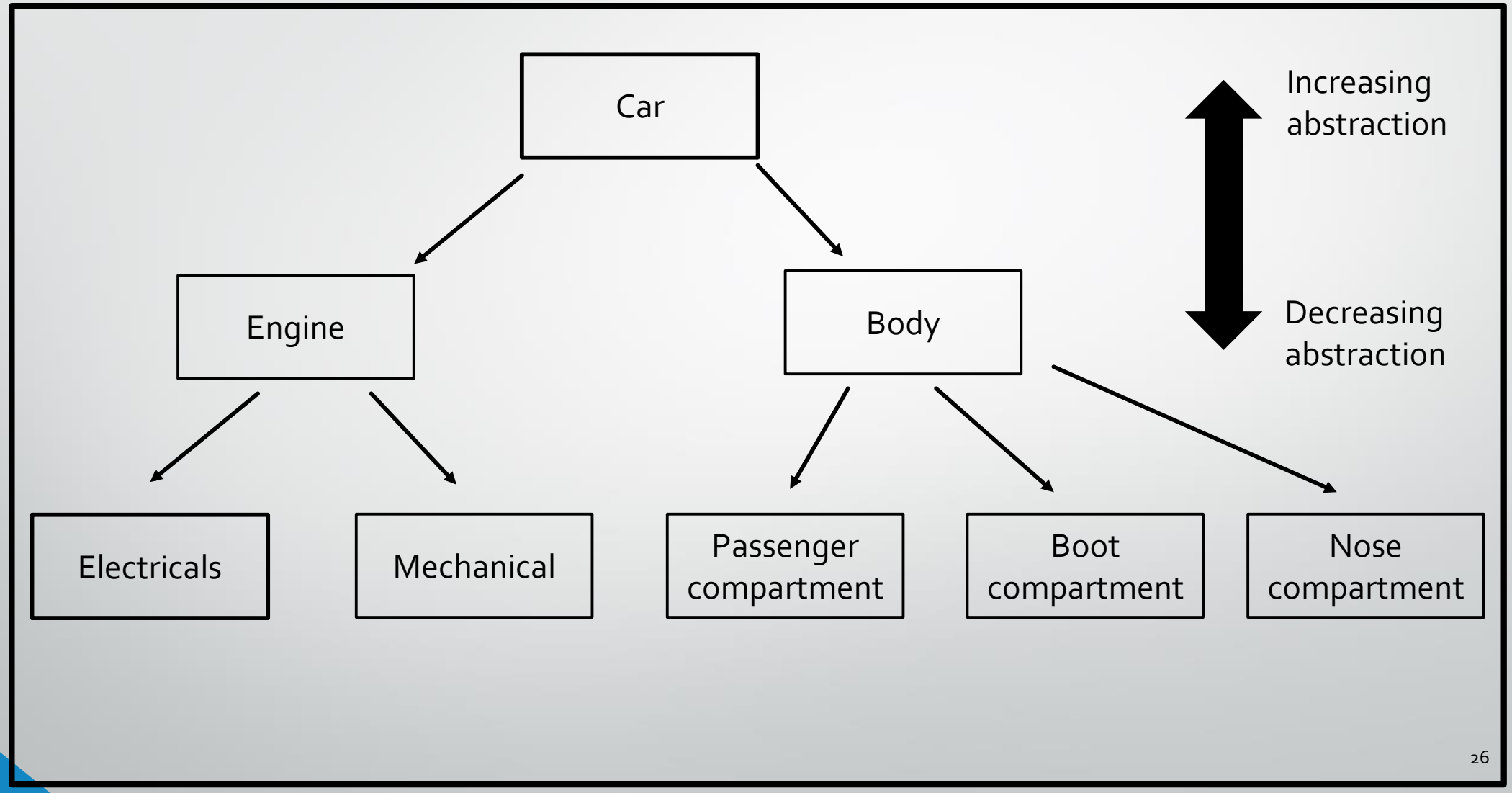

Fig 3. Modularity demonstrated using a car

3.2.4 Hierarchy

- Booch et al. (2007) define hierarchy as simply an ordering of abstractions. By arranging the abstractions in order the development of the system is made much easier and the problem can be understood much better.
- Using hierarchy the system can be arranged in order of increasing abstraction.
- Hierarchies can be described as singular inheritance (ISA) or multiple inheritance. They can also be described as an object structure (part –of relationship).
- In single inheritance (ISA) an entity inherits from one parent while in multiple inheritance an entity inherits from multiple parents. The object structure shall be discussed in details in a later lesson.
- Fig 4 describes a simple single inheritance relationship in order of abstraction. Vehicles can be either cars, pickups or lorries. Cars can be either sedans/saloons, hatchbacks or station wagons. The order of abstraction is also shown in the fig 4; decreasing levels of abstraction going down the hierarchy and vice versa. Note the change in direction of the arrows as compared to fig 3. This is the correct notation to use when describing inheritance and will be emphasized on when describing the class diagram in a later lesson.
- Multiple inheritance occurs when an entity inherits from more than one parent and will be discussed together with object structures in a later lesson.

3.2.4 Hierarchy

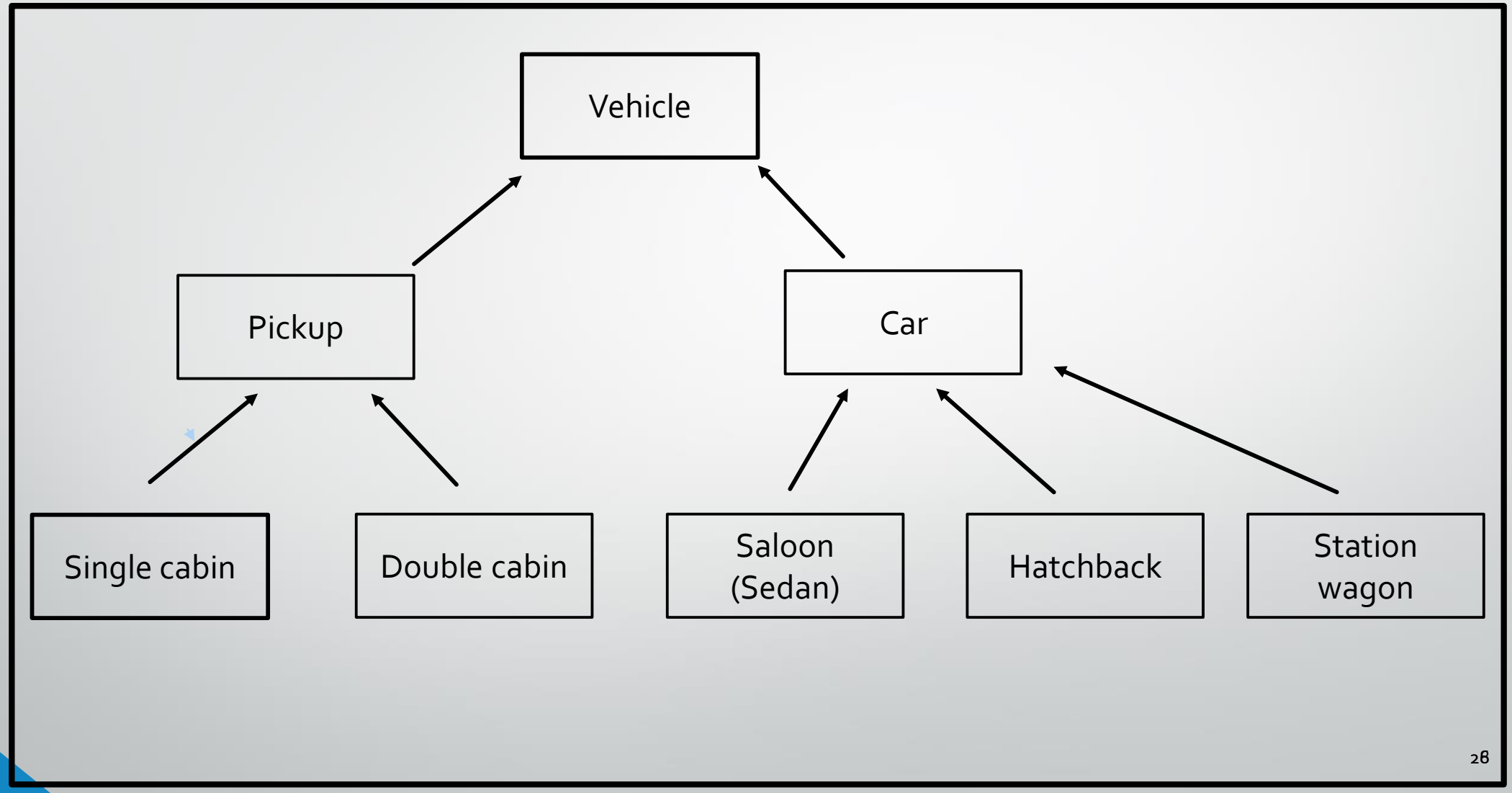

Fig 4 Single inheritance hierarchy (ISA)

Part 4

Object Orientation concepts

4.1 Introduction

- So far we have examined the foundations of object orientation. There are certain terms which have also come up together with others; these concepts together with the foundation concepts make up what is referred to as the object model in OO. The object model enables visualization of the system to be developed using objects.
- The concepts that are described in this part are:
  - Object
  - Class
  - Attribute
  - Operation
  - Interface (Polymorphism)
  - Component
  - Package
  - Subsystem

4.2 Object

- As described earlier object orientation is based on the concept of real world objects. This means that a system is developed around the concept of an existing entity (object). How then do we describe an object?
- An object is an entity or thing that exists in the real world and can be called by name. This means that an object is something that can be perceived by human senses. This does not mean that an object must be physical; it can be a process (meaning it has been perceived to happen) as well.
- Examples of objects include cars, people, software, genes (and chromosomes), lakes, oceans, houses, schools, and so on.
- Every object will have the following characteristics:
- Identity: every object has a unique identity; no two objects are the same.
- Behavior: these are the things (actions or verbs if you like) that the object can do.
- State: this refers to the properties of the object at that particular time. This of course means that the state of an object can change at different times (though not all objects).

4.2 Object (cont'd)

- Having learnt this let us look at two examples of objects.
- First a person like me. I am John M (identity) and I can run, swim, give lectures and so on (my behavior). I am 75 kgs light (I wish) and this is my state at this this time.
- Let us consider a pet dog called Max (identity). Max can bite, wag his tail, and bark (his behavior). He is fluffy (state).
- Clearly this isn't enough to wholly describe everything about our object; we shall get more descriptive shortly. However, point to take from our definition is that every object has state, identity and behavior. If it is lacking even one of the three then it ceases to be an object.

4.3 Class

- A class is a grouping name for all objects that share the same properties, behavior, relationships and connotations. A class therefore can be described as a blueprint for a certain group of objects. This means that for the object to belong to that class it must have the properties described for the class.
- By definition an object is an instance of a class. From our knowledge of English we know that an instance is simply an example or a single occurrence.
- A class is an abstraction in that we only take account of the properties that are of interest and suppress the others.
- For example supposing we declare a class called dog. The properties of the dog of interest are breed, shoulder height and color. The behavior of interest is its top speed and how high it can jump. From this example you notice that there are many properties and behavior we have not bothered with. For instance we could have noted the age of the dog (not interested) or even the average training period (again not interested).
- Let us add a few more notations using an example closer to home like a course.

4.3 Class (cont'd)

- The UML notations for both the class and the object shall be described in a later lesson; by way of introduction both of them are described in a rectangle. Now back to our example.
- We can now describe our class as follows:
- Class name: Course
- Properties: Course Code, Name, YearOffered, CreditHours, Type
- Behavior: Add student, delete student, print class list, enter student marks
- Based on these details we can choose to instantiate the class.
- Remember the class is just a blueprint from which to create instances (members) of the class with their own unique identities and states corresponding to the description of the class.

4.4 Attribute

- An attribute is a desirable feature used to describe members of a class.
- For example in the Course class the attributes are Course Code, CourseName, YearOffered, CreditHours, Type
- We can create 3 objects with the attributes we desire:
- Object 1: HUF310 (Course Code), Object oriented analysis and design (name), Year 3 (offered in third year), 45 (credit hours), Core (it is a core course, as opposed to elective)
- Object 2: HUF202 (Course Code), Network design (name), Year 2 (offered second year), 45 (credit hours), Core (it is a core course)
- Object 3: HUF 101, Communication skills, Year 1, 45, Common unit

4.5 Operation

- The operations of a class are the different "actions" or behaviors attributed to it. This means these are the things that all members of the class can do.
- Using our Course class the operations of members of the class are:
- Add student, delete student, print class list, enter student marks
- This means that any instance of the class (object) can perform any operations allowed (declared) for the class.
- If we were to declare a dog class with Max being a member of the class then the operations for the class would be bite, wag tail, and bark. Therefore all the members of the dog class can perform these three operations.
- Suppose you were to declare a class called "Mobile phone". What do you think would be the attributes and operations of this class?

4.6 Interface (Polymorphism)

- Polymorphism literally means "many shapes". In object orientation the term is used to refer to when objects belonging to different classes implement a behavior (operation) differently.
- As can be seen polymorphism is closely related to inheritance since this is where different implementations of the same operation is easily seen and implemented.
- A classical example of this is the use of the "+" operator in object oriented programming (don't worry if you haven't done programming yet, this explanation is simple). When two integers (numbers) invoke this operator the result is the normal addition of numbers. For example, 5 + 2 = 7. However, when the two numbers are declared strings then the "+" concatenates the two strings. For example in this case 5 + 2 = 52. When it is actually two or more strings then we have "none" + "the" + "less" = nonetheless.
- Another example is yourself. When you are here listening to the lecture you are a student. After the session you go home and become son/daughter/father/mother as the case maybe. These are different implementations of yourself.
- One other good example is provided by Ashrafi (2013). As a worker you "work" (operation). However, different workers implement work operation differently. A mechanic, chef, policeman and professor all implement the operation "work" differently.

4.6 Interface (Polymorphism) (cont'd)

- Interfaces also implement polymorphism. This is seen mostly when writing programs using object oriented languages.
- For example one can declare an interface and call it shape (this is also a classical example found in many texts and online). Depending on the shape declared it will implement the operations of the interface "shape" differently. If there is an operation called area, a circle will use a different formula from a square.

4.7 Component

- A component is defined as a representation of a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. (UML 2.0)
- Consequently this means that it acts as a black box whose external behavior is completely defined by its provided and required interfaces.
- Components are usually used to visualize the physical parts of the system such as files and libraries.
- Components will be described in greater detail in a later lesson; at this point we simply introduce the concept.

4.8 Package

- "A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management." (tutorialspoint.com)
- Packages can be used to:
- Provide an encapsulated namespace within which all names must be unique;
- Group semantically related elements;
- Define a "semantic boundary" within the model;
- Provide units for parallel working and configuration management in design. (Arlow and Neustadt, 2013)

4.9 Subsystem

- "In UML models, subsystems are a type of stereotyped component that represent independent, behavioral units in a system. Subsystems are used in class, component, and use-case diagrams to represent large-scale components in the system that you are modeling.
- You can model an entire system as a hierarchy of subsystems. You can also define the behavior that each subsystem represents by specifying interfaces to the subsystems and the operations that support the interfaces.
- In diagrams, compartments display information about the attributes, operations, provided interfaces, required interfaces, realizations, and internal structure of the subsystem.
- Typically, a subsystem has a name that describes its contents and role in the system." (ibm.com)
- A subsystem can also be simply defined as a combination of a package and a class. Consequently this means that a subsystem can also have instances.

Part 5

Strengths and weaknesses of Object Oriented Paradigm

5.1 Strengths

- Since the paradigm is based on real world entities it makes it that much easier for designers to come up with the classes for the design. As discussed earlier if a student is an entity of interest in the system then the designer will simply build a student class and describe the attributes and behavior of interest to the student class.
- Reuse is one of the key pillars of object oriented design; the same software can be reused to solve similar problems in the domain. Further, inheritance can also be used to extend functionality to developed classes. This has the overall effect of increasing productivity across the board.
- With the same principles in mind object oriented systems can accommodate change. For example changes can be made to a class without having to change all the code in the program.
- Because of the foundations of object orientation namely encapsulation, ability to isolate code (classes), and use of modularity the risk element is vastly reduced during system development.
- Other concepts such as polymorphism also introduce flexibility in the whole design.

5.2 Weaknesses

- The many layers of software introduce an element of overheads to the system.
- There are memory access challenges with some systems due to the complexity of the software in terms of the many objects that need to interact with each other in different and complex ways.
- For many programmers who have learnt earlier methods it is sometimes a steep learning curve to switch to object oriented methods.

Part 6

Application areas

Application areas of OO

- At this point you may be wondering…where do we apply object oriented technologies (programs and systems) in industry today?
- There are many areas where OO is applied and quickstart.com offer some good insight into these areas:
- Client – server systems – these are systems used over networks where a server hosts the applications and services needed by clients over the network. The clients connect to the server to access the services and applications. Clients include desktops, tablets, mobile devices and so on. Object oriented client server internet (OCSI) provide such applications; in such a setup there is a client, server, and object oriented programming.
- Object oriented databases – this is the technology mostly being used in database systems today. The database is built around objects rather than data. The objects contain attributes and methods (operations) just as described earlier in the chapter. They are also called object database management systems (ODBMS).
- Simulation and modeling systems – object oriented technology has simplified the design of these complex systems especially in areas like science and medicine.
- Real time system design – these are systems that provide solutions in real time, such as those found in aircrafts. Object oriented methods make it easier to design these systems.

Application of OO (cont'd)

- Hypertext and hypermedia – ever done some basic HTML code writing? If not then hypertext refers to media that "opens" other texts or media. What this means is that you click on a link (the text) and it opens another text or media. In essence the hypertext is a pointer to other text. The same concept applies to hypermedia; it opens media such as sound or a movie and so on. These technologies also utilize object oriented technology.
- Neural networking – this is an area of artificial intelligence (AI) that utilizes object oriented programming to simplify the process.
- AI expert systems – these are written mostly in Python which is an object oriented programming language.
- CAD (Computer Aided Design) systems – these are systems used by engineers in design and manufacturing environments. Object oriented tools are used to increase the accuracy of flowcharts and blueprints used to develop solutions.

Summary

- The object oriented paradigm can be traced as follows: procedural paradigm → modular (structured) paradigm → data abstraction paradigm → object oriented paradigm
- The foundations (also referred to as the pillars) of object orientation are abstraction, encapsulation, modularity, and hierarchy.
- Object oriented concepts introduced in this chapter were object, class, attribute, operation, polymorphism, component, package and subsystem.
- The strengths of object orientation are based on the foundations; reuse, flexibility and scalability are also key strengths.
- Weaknesses of object orientation are based on the supposed inefficiency and complexity of the process, especially for those schooled in other methods.
- There are several applications of object orientation in fields as diverse as science and medicine, to office automation tools.

References (1 of 2)

- Arlow, J., & Neustadt, I. (2013). *Uml 2 and the unified process: Practical object-oriented analysis and Design* (Second). Addison-Wesley.
- Ashrafi, N., & Ashrafi, H. (2014). *Object Oriented Systems Analysis and Design* (1st ed.). Pearson.
- Bachmann, K.-H. (1966). Dijkstra, E. W.: A Primer of ALGOL 60 programming, A. P. I. C. Studies in date processing, no. 2, Academic Press, London, New York 1962. XI + 114 Seiten. Preis 30 s. *Biometrische Zeitschrift*, *8*(1-2), 127–127. https://doi.org/10.1002/bimj.19660080123
- Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Conallen, J., & Houston, K. A. (2007). *Object-oriented analysis and design with applications* (Third). Addison-Wesley.
- Geeks for Geeks. (2022, July 7). *Introduction of programming paradigms*. GeeksforGeeks. Retrieved September 13, 2022, from https://www.geeksforgeeks.org/introduction-of-programming-paradigms/
- Hernandez, B. (2022, March 21). *10 applications of Object Oriented Programming*. 10 Applications of Object Oriented Programming. Retrieved September 13, 2022, from https://www.quickstart.com/blog/10-applications-of-object-oriented-programming/
- IBM Corporation. (n.d.). *Subsystems*. Rational Software Modeler 7.5.0 : Subsystems. Retrieved September 13, 2022, from https://www.ibm.com/docs/en/rsm/7.5.0?topic=diagrams-subsystems
- Morkar, V. (2021, April 20). *What is the difference between structured and Object Oriented Programming*. Medium. Retrieved September 13, 2022, from https://vedant-morkar19.medium.com/what-is-the-difference-between-structured-and-object-oriented-programming-215ff563ca98
- Ojo, A., & Estevez, E. (2005). (rep.). *Object-Oriented Analysis and Design with UML - Training Course* (Vol. 1).

References (2 of 2)

- Statista Research Department. (2022, April). *Internet users in the world 2022*. Statista. Retrieved September 13, 2022, from https://www.statista.com/statistics/617136/digital-population-worldwide/
- *Structured Programming - IT236 Course*. Structured Programming. (n.d.). Retrieved September 13, 2022, from https://condor.depaul.edu/sjost/it236/documents/structured.htm
- Tiny. (n.d.). *Modular Programming: Definitions, benefits, and predictions*. Blueprint - Blog by Tiny. Retrieved September 13, 2022, from https://www.tiny.cloud/blog/modular-programming-principle/#:~:text=Modular%20programming%20(also%20referred%20to,single%20aspect%20of%20the%20functionality.
- Tutorials Point. (n.d.). *UML - component diagrams*. Tutorials Point. Retrieved September 13, 2022, from https://www.tutorialspoint.com/uml/uml_component_diagram.htm?key=package%2Bdiagram
- Tutorials Point. (n.d.). *Java - packages*. Tutorials Point. Retrieved September 13, 2022, from https://www.tutorialspoint.com/java/java_packages.htm
- University of Cambridge and the Raspberry Pi Foundation. (n.d.). *The procedural paradigm*. Isaac Computer Science. Retrieved September 13, 2022, from https://isaaccomputerscience.org/concepts/prog_pas_paradigm?examBoard=all&stage=all
- University of Cambridge and the Raspberry Pi Foundation. (n.d.). *The OOP Paradigm*. Isaac Computer Science. Retrieved September 13, 2022, from https://isaaccomputerscience.org/concepts/prog_oop_paradigm?examBoard=all&stage=all