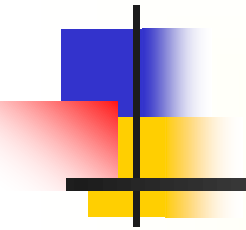


SWENG3101

Object Oriented Programming

Software Engineering Department

AASTU

Chapter 1

Introduction to Object Oriented Programming

Objectives

- At the end of this chapter you should be able to:
  - Discuss and contrast the issues, features, design and concepts of programming paradigms
  - Choose and apply a suitable programming paradigm and language to solve a given problem.
  - Understand basic terms of OOP

Outline

- Program Development
- Programming Paradigms
- Overview of OO principles

Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution
- These activities are not purely linear – they overlap and interact

Cont'd

- To write a program for a computer to follow, we must go through a two-phase process: **problem solving** and **implementation.**

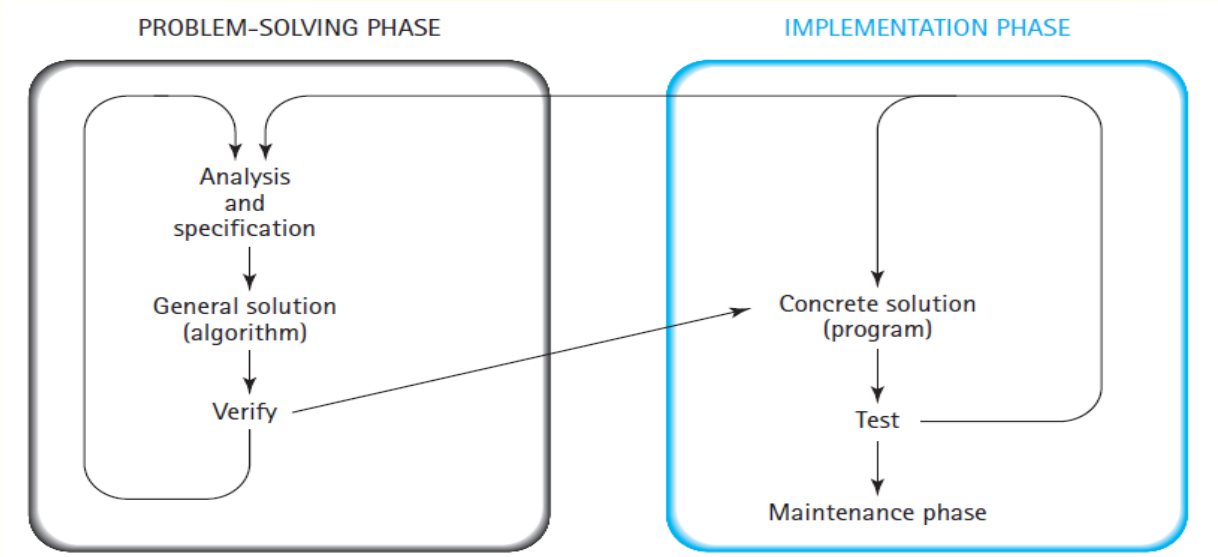
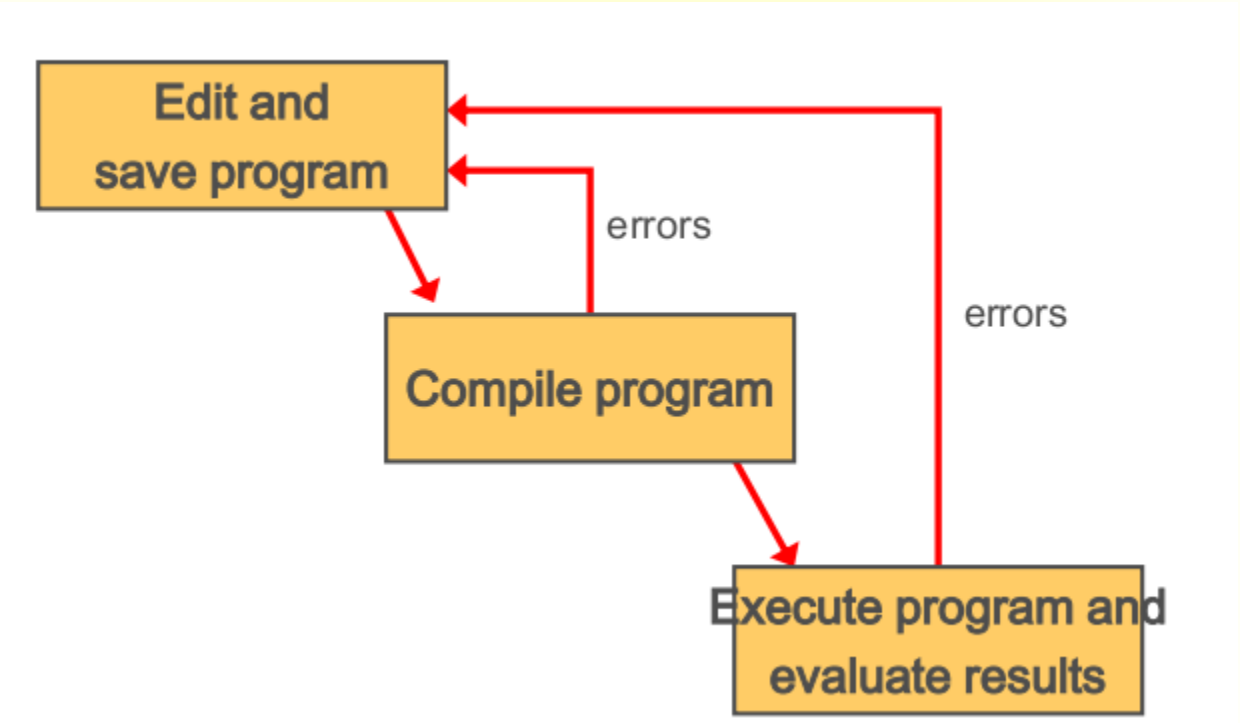


Figure *Programming process*

Program Development

- The mechanics of developing a program include several activities
  - writing the program in a specific programming language (such as java)
  - translating the program into a form that the computer can execute
  - investigating and fixing various types of errors that can occur
- Software tools can be used to help with all parts of this process

Program Development(cont'd)



Programming Languages

- A programming language specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid program statements
- A program must be translated into machine language before it can be executed
- A compiler is a software tool which translates source code into a specific target language
  - Often, that target language is the machine language for a particular CPU type. The Java approach is somewhat different

Programming Paradigms

- Programming paradigms are the result of people's ideas about how programs should be constructed
- A programming paradigm can be understood as an **abstraction of a computer system**
- Different problems are better suited to different paradigms
- In reality, very few languages are "pure"
  - Most combine features of different paradigms
- Main programming paradigms
  - Imperative(procedural) paradigm
  - Functional paradigm
  - Object-oriented paradigm
  - Logical paradigm

Imperative paradigm

- Programming with an explicit sequence of commands that update state
- Specifying the **steps** the program must take to reach the desired **state**.
  - First **do this** and next **do that**
- E.g. Fortran(FORmula TRANslator), Cobol(COmmon Business Oriented Language), Pascal, Ada, C

Functional Paradigm

- It emphasizes the **application of functions**
- Suitable for mathematical computations
- **E.g.** LISP, Haskell
- E.g. Calculate the sum of an array of a list of numbers

Functional

```
Sum = foldr(+)0
```

Imperative

```
int sum(int *arr, int lenght){
int i, sum = 0;
for(i = 0; i < lenght; i++) { //iterate
the loop until the pointer has
reached the end of the array
sum += arr[i];
}
return sum;
}
```

Declarative Programming

- Represents the logic of a computation
- Defining the algorithm in terms of all possible cases without explicitly stating how each case is accessed
- Programs describe **what** the computation should accomplish, rather than **how** it should accomplish it.
- E.g. Prolog, SQL
- Useful for
  - Logic-based problems
- Disadvantage
  - Difficult to see what is actually happening computationally

The Object-Oriented Paradigm

- Represent the problem as a set of interacting objects
- E.g., Smalltalk, Java, C++, etc.
- A *class specifies the format of objects*
- **Send messages between objects to simulate the temporal evolution of a set of real world**
- Useful for:
  - Large complex systems
  - Highly flexible, extensible systems
- Disadvantages
  - Slower
  - Large memory requirements

Overview of OOP

- Everywhere you look in the real world you see **objects** — people, animals, plants, cars, planes, buildings, and computers and so on.
- Humans think in terms of objects. Telephones, houses, traffic lights, microwave ovens and water coolers are just a few more objects.
- Humans learn about existing objects by studying their attributes and observing their behaviors.
- Programs are composed of lots of interacting software objects
  - Have **properties** (**attributes**): describe the state/data of an object at a certain time
    - e.g., size, shape, color and weight
  - Exhibit **behaviors**(**Method**)
    - e.g., a ball rolls, bounces, inflates and deflates; a baby cries, sleeps, crawls, walks and blinks; a car accelerates, brakes and turns

Overview of OOP(cont'd)

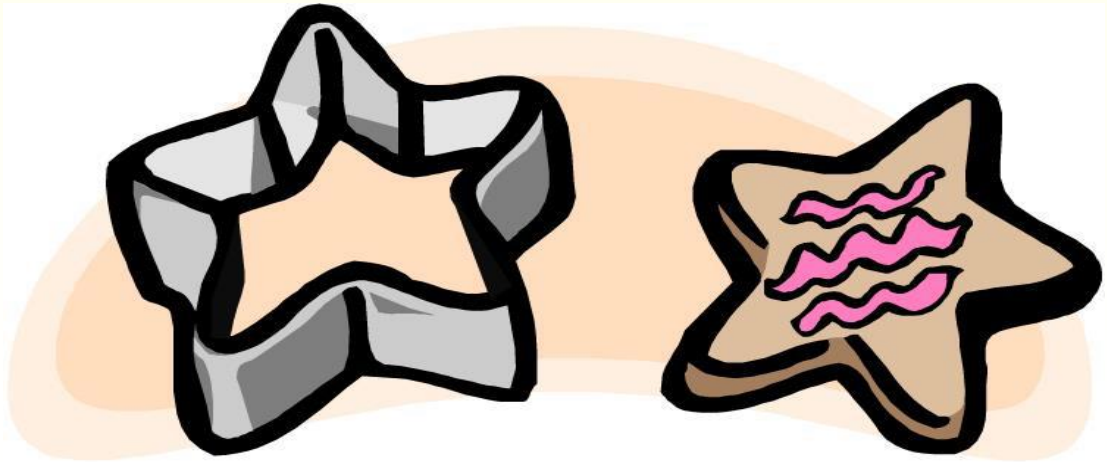
- Different objects **can have similar attributes and can exhibit similar behaviors.**
- A program written in object-oriented style will consist of **interacting objects**.
  - E.g. 1 For a program to keep track of student residents of a college dormitory, we may have many Student, Room, and Floor objects.
  - E.g. 2 For program to keep track of customers and inventory for a bicycle shop, we may have Customer, Bicycle, and many other types of objects.

OOP Principles

- In object oriented programming, the main part in a program are classes, rather than procedures.
- The object-oriented approach lets (allows) you create ***classes*** and ***objects*** that model real world objects.
- **Object**:
  - Basic units of object oriented programming
  - An object can be considered a "*thing*" that can perform a set of related activities.
  - The set of activities that the object performs defines the object's behavior. **For example**, the Hand (object) can grip something, or a *Student* (object) can give their name or address.
  - An instance of a class

OOP Principles(cont'd)

- **Class:**
  - A *class* is simply a representation of a type of *object*.
  - A collection of objects of similar types.
  - Set of objects that share common structure and behavior.
  - It is the blueprint, or plan, or template, that describes the details of an *object*.
  - *Class* is composed of three things: a name, attributes, and operations.
  - class is like a cookie cutter



OOP Principles(cont'd)

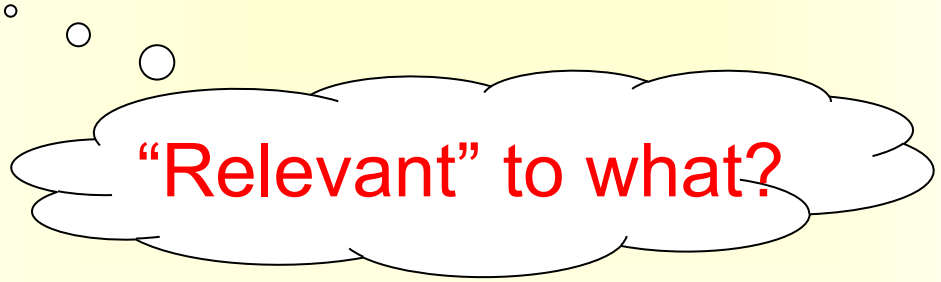
- Example
- **Class**: Phone
  - **attributes:** Type, Name, Model type, Size, Screen color,…
  - **Method**: make-call (), Receive-call (), Send-message (), Receive-message (), Take-picture (), …

OOP Principles(cont'd)

- Four concepts are critical to understanding object models:
  - Data abstraction
  - Polymorphism
  - Inheritance
  - Encapsulation
- These can be remembered with the acronym "***A Pie***," for ***A***bstraction, ***P***olymorphism, ***I***nheritance, ***E***ncapsulation.

Data- abstraction

- principle of ignoring those aspects of an object that are **irrelevant to concentrate on those that is relevant.**



- ... relevant to the given project
- Focusing on the essential features and ignoring the non-essential ones.
- Abstraction separates objects from the extra data and behavior that doesn’t directly belong there.
- Example: In university, if a student asked address might say ***Woreda 02, House No:xxxx, AA;*** *(wouldn’t tell the number of rooms, color of walls, seats,…)*

Encapsulation

- *Encapsulation is the mechanism that binds together code and the data it manipulates,* and keeps both safe from outside interference and misuse.
- Wraps attributes and **operations** (behaviors) into objects — an object's attributes and operations are intimately tied together.
- It is the insulation of data from direct access.
- Objects have the property of **information hiding**.
- Objects may know how to communicate with one another across well-defined **interfaces**, but normally they are not allowed to know how other objects are implemented.
  - implementation details are hidden within the objects themselves.

Encapsulation (cont'd)

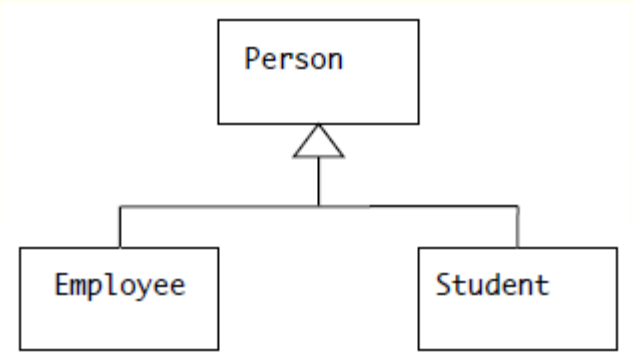
- E.g 1. We can drive a car effectively without knowing the details of how engines, transmissions, brakes and exhaust systems work internally—as long as we know how to use the accelerator pedal, the brake pedal, the wheel and so on.
- E.g. 2. Encapsulation allows us to consider what a light switch does, and how we operate it, without needing to worry about the technical detail of how it actually works.

Inheritance

- The process by which objects of one class acquire the properties of objects of another class.
- An object oriented system organizes classes in to a sub-class, super -class hierarchy. At the top of the hierarchy are the most general classes and at the bottom are the most specific one.
- Sub class inherits all of the properties (non-private attributers) and methods defined in its super class.
- Inheritance provides the idea of reusability (allowing objects to be built from other objects).
- Inheritance represents the *is a* relationship between data types. For example: A sailboat is a boat.

Inheritance(cont'd)

- Example: Consider people at AASTU. Broadly speaking, they fall into two categories: **employees and students**.
  - 1. There are some features that both employees and students have in common - whether a person is an employee or a student, he or she has a *name, address, date of birth, etc*.
  - 2. There are also some features that are unique to each kind of person - e.g. an *employee has a pay rate*, but a student does not; a *student has GPA*, but an employee does not, etc.



"is –a" relationship

Polymorphism

- *Polymorphism (from the Greek, meaning "many forms")*
- It is a feature that allows one interface to be used for a general class of actions.
- Polymorphism mainly requires the ability for different objects to have methods with the same name, but different parameters.
- Any of the objects referenced by this name is able to respond to some state of operations in a different way.
- Using functions in different ways depending on what they are operating on (function overloading) is polymorphism,
- one thing with several distinct forms.
- E.g. Your mobile phone, one name but many forms as phone, as camera, as mp3 player, as radio

Polymorphism (cont'd)

- **Analogy**, a dog's sense of smell is polymorphic.
  - If the dog smells a cat, it will bark and run after it.
  - If the dog smells its food, it will salivate and run to its bowl.
  - **The same sense of smell is at work in both situations**. The difference is what is being smelled, that is, the type of data being operated upon by the dog's nose!
- *What is the advantage?*

Dynamic binding

- The process of determining dynamically which function is to be involved (i.e. at runtime).
- Making this determination at compile time is called **static binding**.
- **Overloaded methods are resolved at compile time** based on the arguments supplied.
- **Overridden methods are resolved at run time**.
- If a subclass methods **has the same name, parameter list and return types as super class method**, it is said that the sub-class method **overrides the super class method**

Activity

- Identify objects, attributes and methods
  - Student Information Management System
  - Hotel Reservation System
  - Online Shopping System