20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Karaka Sri Sai Nitin
CB.EN.U4CYS21027
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

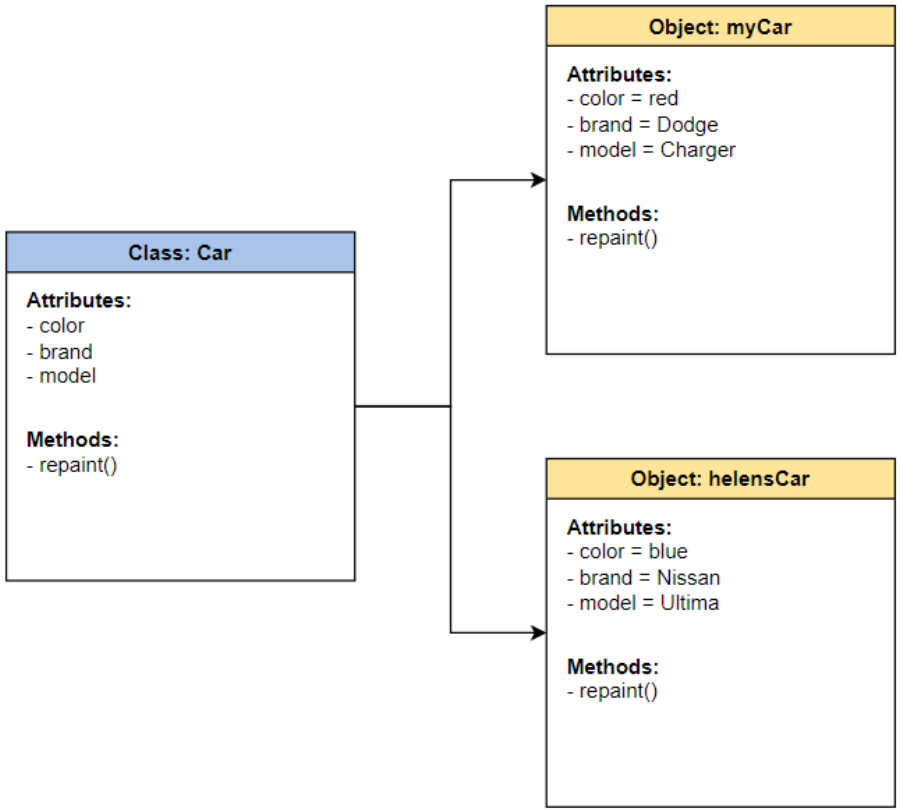
AMRITA
VISHWA VIDYAPEETHAM

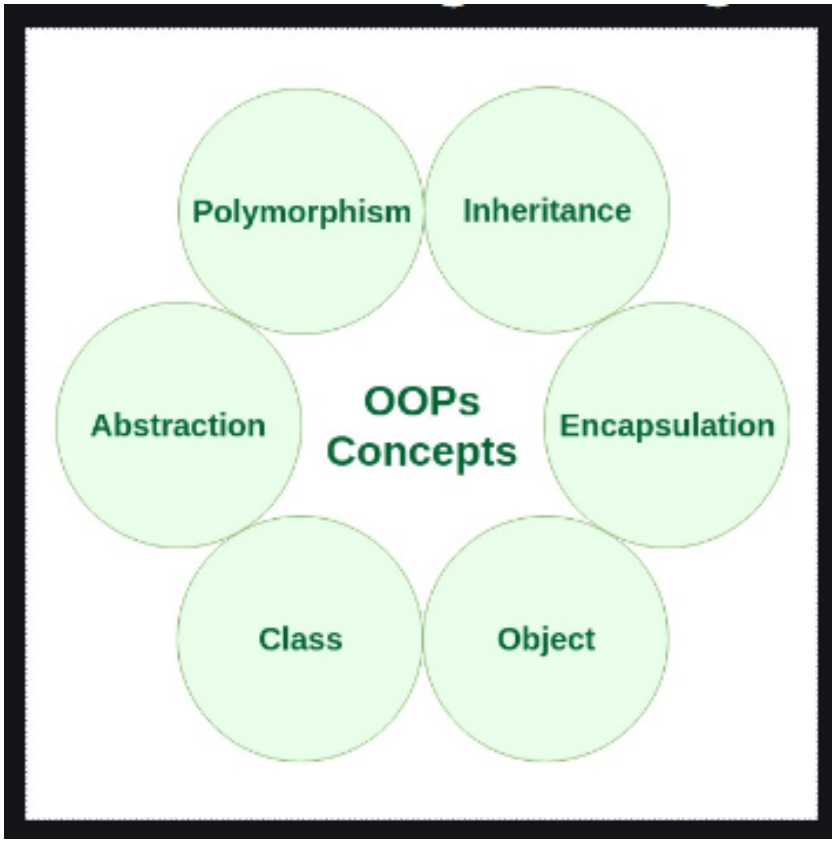
Outline

What is Object - Oriented Paradigm?

- **Object Oriented Paradigm**,also known in short as OOP is defined as programming paradigm which is built on the concept of object and classes.
- It is employed to organise a software application into easily **reusable, basic code blueprints** known as classes that are utilised to generate unique instances of **objects**.
- **Object-Oriented Paradigm** approach identifies **classes of objects** that are closely related to the methods with which they are associated.
- **Class** is a user-defined data type that can be accessed and used by creating an instance of that class. It has its own data members and member functions.(Blueprint for creating objects).
- An **Object** refers to an instance of a class. Its a physical entity, which has no state or behaviour.
- **Advantages of OOP**: 1. **Increase productivity in software development** 2. **Makes troubleshooting simpler** 3. **Reinforces security** 4. **Provides design advantages** 5. **Lowers development costs**. 6. **Code Flexibility** through polymorphism.

Structure of Object Oriented Paradigm





- Uses of **OOP** is that it provides 1. **Modularity** to the code which makes the code easier to understand, maintain and update it. 2. It also facilitates **code reuse** by the creation of **objects and classes** by creating multiple instances of them.

Features and Implementation of OOP

Object Oriented Paradigm is implemented with a set of key concepts and features:

- **Classes** are templates that define the structure and behaviour of objects while **Objects** are instances of classes created at runtime and has its own set of attributes.
- **Encapsulation** is bundling of data and methods that operates on data within a class/object and helps in hiding internal details of an object.
- **Abstraction** allows the programmer to focus on essential features and hiding the irrelevant details. It creates abstract classes and interfaces which contain attributes and methods without implementation.
- **Inheritance** refers to the process of gaining properties where derived class can inherit from base class. It allows code reuse and provides parent-child relation.
- **Polymorphism** enables a single interface to represent various type of objects, achieved through method overriding. **Compile Time Polymorphism** and **Runtime Polymorphism** are two different types of polymorphism.

Code Example of Implementation of OOP in Common Languages

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side
```

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def make_sound(self):
        pass

#Inheritance
class Dog(Animal):
    def make_sound(self):
        return "Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow!"

#Polymorphism
def animal_sound(animal):
    return animal.make_sound()

dog = Dog("Buddy")
print(animal_sound(dog))
```

Figure: Implementation of OOP Concepts such as **Abstraction, Inheritance and Polymorphism** in common languages i.e. Python in the above code

Introduction to TypeScript

- **TypeScript** is a syntatic superset of **JavaScript** which adds **static typing**, which means that TypeScript adds syntax on top of JS , allowing developers to add **types**.
- **TypeScript** is a multi-paradigm , free and open source high level programming language which was designed by **Microsoft**. It's latest model is the **TypeScript 5.0** which released in 2023.
- **TypeScript is used over JS** because **JS** is a loosely typed language which is difficult to understand what datatypes are used and also function parameters and variables dont have info. Hence, **TS** allows specifying datatypes and has ability to report errors when types dont match.
- **TypeScript** uses **Compile-Time Checking** -> Advantage of **TypeScript!**
- Some of the features of **TS** is that 1.**Static Typing**, 2.**It supports JS Libraries**, 3.**DOM Manipulation**, 4.**Portable Language**.
- **Advantages of TS**: 1. **Namespace Concept Defining a module**, 2.**Strongly Typed or Static Typing**, 3. Great Tools such as **IntelliSense** for code autocompletion, 4.**Supports all kinds of JS Framework such as Angular**.

TypeScript Language using Object Oriented Paradigm(OOP)

TypeScript is closely related to Object-Oriented Programming (OOP) as it supports traditional OOP principles with help of static typing by enhancing JS.
Some of the key features of Object Oriented Programming that align with Typescript is:

- **TypeScript** introduces a class-based syntax for OOP allowing developers to define classes.
- It supports **single and multiple Inheritance** through the use of **extends** keyword allowing class to inherit properties.
- It supports **Encapsulation** by providing access modifiers to control visibility of members.
- It allows creation of **Abstract** classes enabling developers to define blueprint for a class without full implementation.
- It also supports **Polymorphism** through method overriding and overloading enables developers to write methods that work with various objects.
- Hence, **TypeScript** strives to improve code organisation, maintainability, and scalability by implementing OOP principles in real world projects.

Example Code of How TypeScript uses OOP

```typescript
class Animal {
  private name: string;

  //Constructor Implicitly public
  constructor(name: string) {
    this.name = name;
  }

  public getName(): string {
    return this.name;
  }
}

const myAnimal = new Animal("Lion");
console.log("Original Name:", myAnimal.getName());
```

```typescript
abstract class Sensor {
  // Abstraction
  abstract ping(): void;
}

class Sonar extends Sensor {
  // Inheritance
  ping() {
    console.log("Sonar ping!");
  }
}

class Radar extends Sensor {
  // Inheritance
  ping() {
    console.log("Radar ping!");
  }
}

const sonarSensor: Sensor = new Sonar();
const radarSensor: Sensor = new Radar();

// Polymorphism
sonarSensor.ping();
radarSensor.ping();
```

Figure: Implementation of OOP Concepts such as **Encapsulation, Access Modifiers, Abstraction, Inheritance and Polymorphism** in TypeScript

.

What is Concurrent Paradigm?

- **Concurrency** generally refers to events that are happening at the same time.
- **Concurrent Programming** is a technique in which two or more processes start, running through context switching and complete in an overlapping time period by managing access to **shared resources**.
- **Concurrent Programming** happens on a single core of CPU but **Parallel Programming** is achieved by executing two processes on 2 separate CPU cores.
- **Concurrent Programming** is **Orthogonal** as it is independent of underlying prog paradigm allowing seamlessly applied to object-oriented or functional programming.
- **Non-Determinism** is a property of concurrency which shows the unpredictable order of execution and outcomes when multiple tasks run concurrently.
- **Concurrent Programming Primitives** : suite of primitives for coordinating and synchronizing in code such as **Mutex Locks, Semaphores and Condition Variables**.

Features and Uses of Concurrent Programming Paradigm

- Some of the features that led to development of **Concurrent Programming is**:
  Resource utilization is that its more efficient to use the wait time to let another program run.
  Fairness shows us that multiple users and programs have equal claims on machine's resources.
  It is possible to write multiple programs for each single task and coordinate with each other rather than single program for all tasks.
- These methods lead to the development of **Threads** which is a facility to allow multiple activities within a single process.The use of Threads are 1. **To perform async or background processing** , 2. **Increases responsive of GUI apps**, 3. **Advantageous for multiprocessor systems**
- Some of the Uses of **Concurrent Paradigm** are:
  1. It is crucial in handling multiple tasks in real time systems with strict timing requirement.
  2. It is essential for server-side apps to handle multiple client requests simultaneously.
  3. It is used in asynchronous programming which manages non blocking I/O operations which handle multiple requests simultaneously.

Some Types of Concurrent Programming

- Some of the types of **Concurrent Programming** are:
- **Process Based Concurrency**: Involves execution of independent and separate processes within its memory space(communicate through Inter-Process Communication).
- **Thread Based Concurrency**: It utilizes threads that are lightweight, independent units of execution which share same memory space, making it more efficient.
- **Coroutine Based Concurrency**: It involves cooperative units of execution known as **co-routines**. It can pause and yield control to others, allowing for sequential code with asynchronous behaviour.

Example Code of Implementation of Concurrency in Common Languages

```python
from time import sleep
from multiprocessing import Process

def task():
    sleep(1)
    print('This is coming from another process', flush=True)

if __name__ == '__main__':
    process = Process(target=task)
    process.start()
    print('Waiting for the new process to finish...')
    process.join()
```

```python
#Thread - Based
from time import sleep
from threading import Thread

def task():
    sleep(1)
    print('This is coming from another thread')

thread = Thread(target=task)
thread.start()
print('Waiting for the new thread to finish...')
thread.join()

#Co-Routine Based
import asyncio
async def custom_coro(message):
    print(message)
asyncio.run(custom_coro('Hi from a coroutine'))
```

Implementation of 3 Types(**Process Based,Thread Based, Coroutine Based**) of Concurrency in Python.

Introduction To Clojure

- **Clojure** is a high level, multi-paradigm , dynamic **functional** programming language.
- It is a **Functional** programming language which treats its computation as evaluation of math functions and avoids mutable data.
- It belongs to the family of **LISP** programming language and inherits its powerful macro system, which allows to modify the language itself and it uses compilers that makes it possible to run on both **Java and .Net** runtime environment.
- **Clojure** uses a **REPL(Read-Eval-Print-Loop)** driven development style which helps developers interact with code, evaluate expressions and see results immediately.
- **Clojure** programming focuses on **Immutability**, which tells us that object cant be modified once it is created. It is simple and concise programming language.
- It has built in support for concurrent programming which has features such as **Atoms, Agents, Refs, Futures, STM** etc making it easier to write concurrent code.
- This language was created by **Rich Hickey** because he wanted **LISP** to have functional programming and designed for concurrency which works on Java platform.

Clojure Language using Concurrency Paradigm

- **Concurrent Programming in Clojure** allows dealing with simultaneous execution of multiple tasks, along with multi-core processors.
- It is designed to make concurrent programming easier by providing **immutable data structures**. Immutability simplifies concurrent programmign by eliminating locks to protect shared mutable state.
- Some of the few features of Clojure's Concurrency Model are:
- **Atoms** provide a way to manage shared mutable state in coordination(**swap! function** is used).
- **Agents** used for async, independent updates to state. They allow for non - blocking, parallel updates.
- **Refs** coordinated via Software Transactional Memory(STM), allowing for complex, multi-step updates.
- **Vars** are mutable variables that are shared across threads and they isolate changes within threads.
- Concurrency in Clojure is based on **transactions** and all these transactions happen with a dosync block where all changes are atomic and isolated.

Example Code of How Clojure uses Concurrent Paradigm

```clojure
(defn long-running-task [n]
  (Thread/sleep 1000) ; Simulate a time-consuming task
  (* n n))

(defn main-function []
  (let [result-1 (future (long-running-task 5))
        result-2 (future (long-running-task 10))]
    (println "Waiting for results...")
    (println "Result 1:" @result-1)
    (println "Result 2:" @result-2)))

(main-function)
```

```clojure
(defn long-running-task [n]
  (Thread/sleep 1000) ; Simulate a time-consuming task
  (* n n))

(defn main-function []
  (let [result-1 (atom nil)
        result-2 (atom nil)]
    (doto
      (Thread. #(reset! result-1 (long-running-task 5)))
      (.start))
    (doto
      (Thread. #(reset! result-2 (long-running-task 10)))
      (.start))
    (Thread/sleep 2000) ; Wait for threads to finish
    (println "Result 1:" @result-1)
    (println "Result 2:" @result-2)))

(main-function)
```

Implementation of Process Based and Thread Based Concurrency in Clojure.

Comparison of Implementation of OOP in Common Languages to that of Specified Languages

- Some of the Comparisons I found between Implementation of OOP in Python and TypeScript is that:
- TypeScript is a **Statistically-typed** language which enforces strong typing, which catches errors during development which leads to more robust and maintainable code while Python is a **Dynamically Typed** language making it more flexible but may result in **runtime errors**.
- **TypeScript** supports class-based inheritance and interfaces, making it more aligned with OOP principles allowing it to create well-defined and structured class. It is much more **consistent and intuitive** than Python even though Python is versatile.
- **TypeScript** has enhanced, rich tooling with great IDE support which makes it a powerful language for large codebases while **Python** simplicity and readability contribute to a better developer experience which can help in faster prototyping.
- **TypeScript** is mostly preferable for large, statistically typed projects in the web-development space in frameworks of Angular and React while **Python** is more flexible and make it useful for wider range of apps.

Comparison of Implementation of Concurrency in Common Languages to that of Specified Languages

- Some of the Comparisons I found between Implementation of Concurrency in Python and Clojure is that:
- **Python** emphasises on relying on multi-threading, multi-processing, or async programming for concurrency while **Clojure** emphasizes immutability, STM and functional programming making it more concurrent friendly model.
- **Python** uses **locks,semaphores** and other mechanisms for managing shared state while **Clojure** uses a more declarative approach, using STM and immutable data structures to manage shared state.
- **Python** considers thread safety and invlove explicit lock mechanisms while **Clojure** designed principles based on immutability and functional programming for more safer concurrency.
- **Clojure** reducers and functional programming gives more support for parallelism without threading.

Real World Projects that use these Paradigms in these Languages

- A real world project that uses **Object Oriented Programming in TypeScript** is **Slack** , which is a cloud-based communication platform that is essentially a messaging app designed for business that aims to revolutionize the way teams communicate and collaborate.Slack's desktop and web clients are extensively built with **TypeScript**, ensuring type safety, maintainability, and scalability.
- A real world project that uses **Concurrency in Clojure** is **Passport Seva Kendra** as PSKs handle various concurrent tasks, including token generation, applicant flow, document verification, and biometric capture. Clojure's concurrency primitives could manage these tasks efficiently, ensuring smooth operations.

References for OOP and TypeScript

1.https://www.tutorialspoint.com/software_architecture_design/object_oriented_paradigm.html
2.https://www.educative.io/blog/object-oriented-programming
3. www.spiceworks.com/tech/devops/articles/object-oriented-programming/
4. https://medium.com/@baruahd5/object-oriented-programming-oops-using-typescript-cdbffd7d0cf5
5. https://blog.appsignal.com/2022/04/06/principles-of-object-oriented-programming-in-typescript.html
6. https://www.javatpoint.com/typescript-features
7. https://birdeatsbug.com/blog/object-oriented-programming-in-typescript
8. https://blog.appsignal.com/2022/04/06/principles-of-object-oriented-programming-in-typescript.html
9. https://slack.engineering/typescript-at-slack/
10. https://en.wikipedia.org/wiki/TypeScript

References for Concurrency and Clojure

1.https://www.educative.io/answers/what-is-concurrent-programming
2.gowthamy.medium.com/concurrent-programming-introduction-1b6eac31aa66
3. https://superfastpython.com/concurrent-programming/#Concurrent_Programming_is_Orthogonal
4. https://en.wikibooks.org/wiki/Learning_Clojure/Concurrent_Programming
5. www.tutorialspoint.com/clojure/clojure_concurrent_programming.htm
6. https://medium.com/helpshift-engineering/simulating-the-passport-seva-kendra-using-clojure-fd88c12dde8c
7. ClojureConcurrencyTalk.pdf
8. https://www.oreilly.com/library/view/clojure-programming/9781449310387/ch04.html
9. https://clojure.org/about/concurrent_programming
10. https://en.wikipedia.org/wiki/Clojure