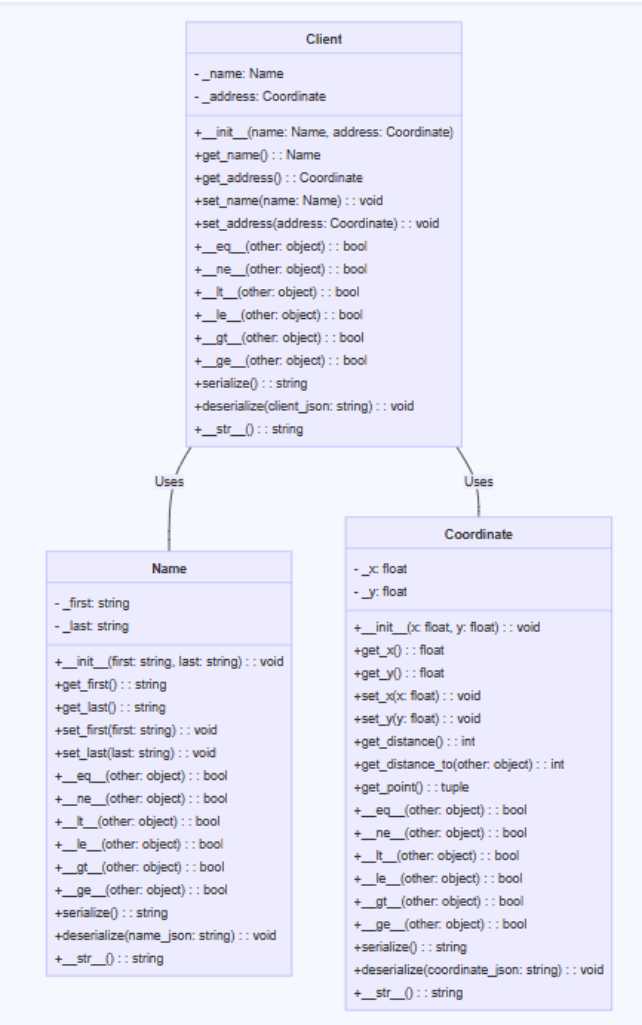
Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías
Ingeniería de Computación

Análisis de Algoritmos (D01)

Integrantes:
Carlos Andres Chico Aguayo
Gael Emiliano Anaya Garcia
Naylea Danae Silva Penaloza
Carlos Humberto Avila Sanchez
Nahomi Itzel Luna Ornelas
Profesor:
Jorge Ernesto Lopez Arce Delgado

26/NOV/2023



La clase **Client** representa a un cliente y contiene dos atributos principales: **name** (nombre) y **address** (dirección), que son instancias de otras clases llamadas **Name** y **Coordinate**, respectivamente. La clase tiene métodos para obtener y establecer tanto el nombre como la dirección del cliente. La relación entre Client y Name es una relación de composición, lo que significa que un Client "tiene un" Name. De manera similar, la relación entre Client y Coordinate indica que un Client "tiene una" Coordinate.

Además, la clase implementa métodos especiales de comparación, como igualdad, desigualdad, menor que, menor o igual que, mayor que y mayor o igual que, basados en la comparación de nombres entre instancias de la clase **Client**. Esto permite comparar objetos **Client** de manera conveniente.

La clase también cuenta con métodos para serializar y deserializar objetos en formato JSON, facilitando la conversión del objeto a una representación de cadena JSON y la reconstrucción del objeto a partir de una cadena JSON.

Adenas hay un método que proporciona una representación en cadena legible para el objeto **Client**, mostrando el nombre y la ubicación del cliente. En resumen, la clase encapsula la información esencial de un cliente y proporciona métodos para manipular, comparar y representar esta información.

Name (Nombre): Representa el nombre de una persona compuesto por un primer nombre y un apellido. No tiene una relación directa con otras clases en este contexto.

El constructor de la clase Name inicializa la instancia con dos parámetros opcionales: el primer nombre y el apellido. Estos valores representan la identidad del individuo. Si no se proporcionan valores, los atributos se establecen en None.

Además, la clase tiene dos atributos principales: uno para almacenar el primer nombre y otro para el apellido. Estos atributos reflejan la identificación nominal de una persona.

Métodos como get_first() y set_last(last) permiten acceder y modificar los valores del primer nombre y el apellido, respectivamente. Estos métodos facilitan la manipulación de la identidad contenida en la instancia.

La clase implementa métodos especiales de comparación. Estos métodos evalúan la relación entre instancias de la clase, centrándose en la comparación de apellidos.

La clase proporciona métodos para convertir el objeto a formato JSON y para reconstruir el objeto desde una cadena JSON. Estos métodos permiten representar la información de la instancia en un formato para después ser utilizado para el intercambio de datos con la interfaz.

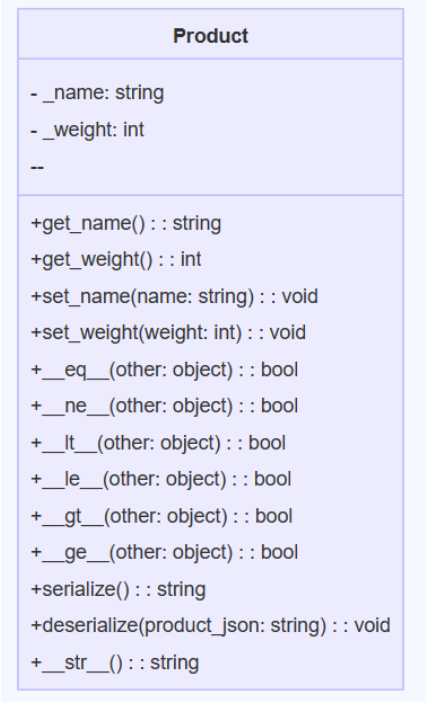
Coordinate (Coordenada): Representa las coordenadas (posición) de algo en un plano bidimensional. No tiene una relación directa con otras clases en este contexto.

El constructor de la clase Coordinate inicializa la posición del punto en el plano cartesiano, con dos parámetros opcionales: x e y. Si no se proporcionan valores, las coordenadas se establecen en None. Además, la clase tiene dos atributos, x e y, que representan las coordenadas del punto en los ejes X e Y, respectivamente.

Hay varios métodos que permiten obtener las coordenadas en los ejes X e Y, respectivamente y otros que permiten establecer nuevas coordenadas, además hay métodos para calcular distancias estos sirven para calcular la distancia del punto al origen (0,0) en el sistema de coordenadas y/o para calcular la distancia entre el punto actual y otro punto especificado.

La clase implementa métodos especiales de comparación. Estos métodos evalúan la relación de la distancia del punto al origen con respecto a otro punto.

En términos prácticos, un Client tiene una instancia de Name y otra de Coordinate. La relación de "uso" indica que Client utiliza instancias de Name y Coordinate para representar información relacionada con el cliente.



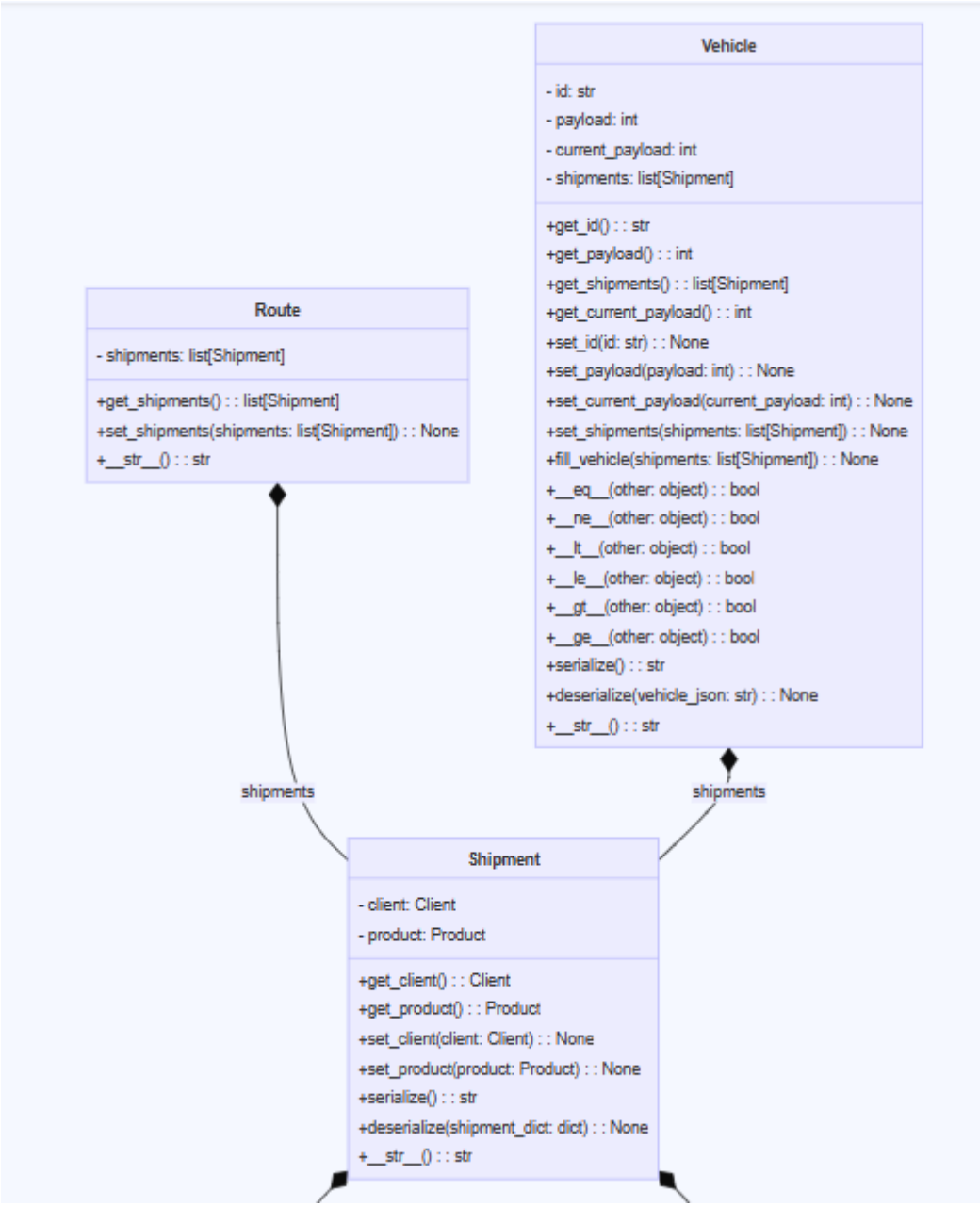
La clase Product es una representación de productos que pueden ser enviados. Tiene dos atributos principales: name (nombre del producto) y weight (peso del producto). Ambos atributos son inicializados en el constructor de la clase. La clase también proporciona métodos para obtener y establecer el nombre y peso del producto.

Además, la clase implementa varios métodos especiales para comparar productos en función de su peso. Estos métodos permiten comparaciones entre instancias de la clase Product.

La clase también incluye métodos para serializar y deserializar objetos en formato JSON. El método serialize convierte un objeto Product en un diccionario y luego lo serializa en formato JSON. El método deserialize toma un objeto JSON y lo convierte de nuevo en un objeto Product.

Finalmente, el método str proporciona una representación en cadena del objeto Product, mostrando el nombre y el peso del producto.

La clase Product encapsula la información sobre un producto, proporciona métodos para manipular esa información, y permite la comparación y la conversión a/desde formato JSON.



La clase Route tiene una relación con la clase Shipment ya que contiene una lista de envíos además la clase Vehicle también tiene una relación con la clase Shipment ya que contiene una lista de envíos asignados al vehículo.

No hay una relación directa entre Route y Vehicle en los atributos de las clases, pero conceptualmente, un conjunto de vehículos puede utilizarse para completar una ruta, y cada vehículo puede tener asignados un conjunto de envíos que forman parte de la ruta.

- **Shipment:** Representa la información sobre un envío específico, incluyendo detalles sobre el cliente y el producto asociados al envío.

  La clase Shipment representa un envío y tiene atributos relacionados con el cliente (Client) y el producto (Product). Un envío puede estar asociado a un cliente y a un producto.

  La clase Shipment tiene métodos para obtener y establecer el cliente y el producto, serializar y deserializar el objeto, y una representación en forma de cadena.

- **Route:** Representa una colección de envíos organizados para formar una ruta de entrega. Puede contener múltiples envíos y se utiliza para gestionar y organizar la distribución de productos.

  La clase Route representa una ruta y contiene una lista de envíos (Shipment). Puede tener varios envíos organizados en una lista.

  Tiene métodos para obtener y establecer la lista de envíos, así como una representación en forma de cadena. Su propósito principal es organizar y gestionar los envíos en una ruta.

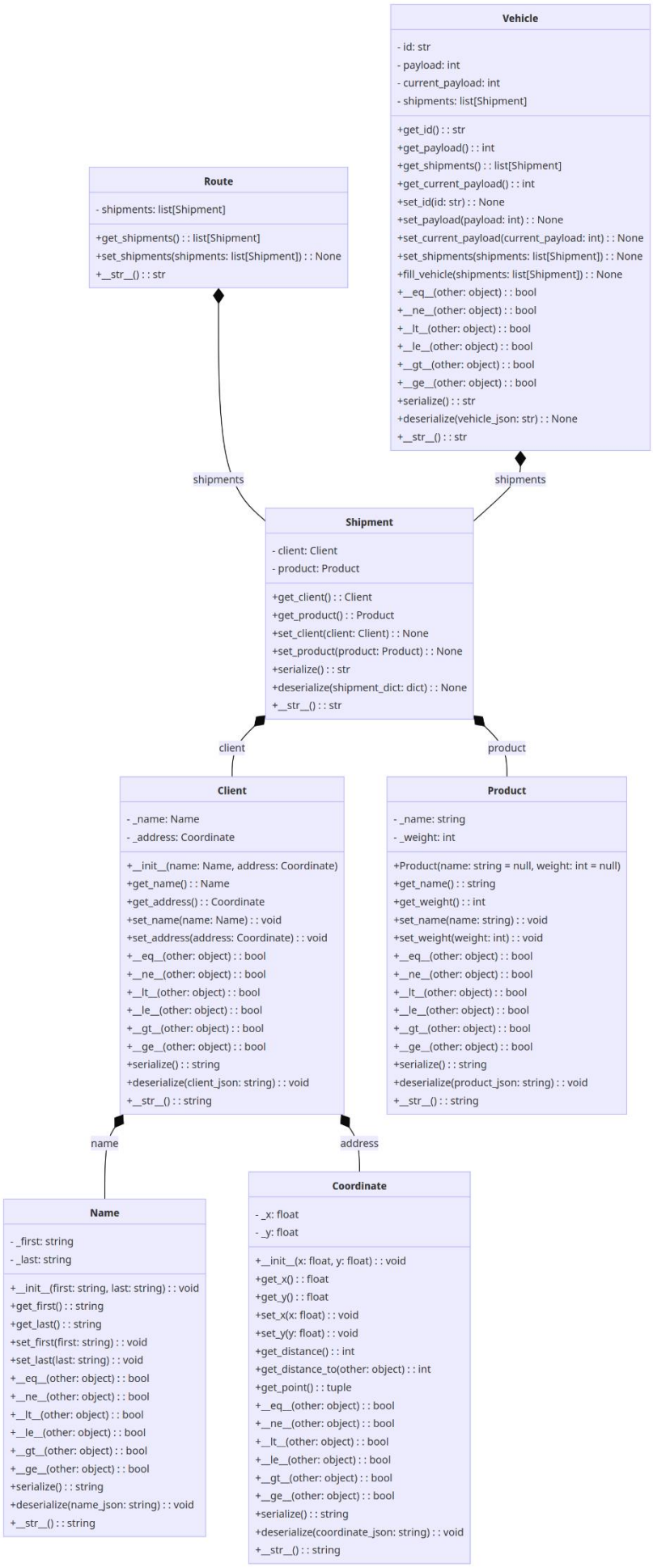
- **Vehicle:** Representa un vehículo que se utilizará para transportar los productos a lo largo de la ruta. Tiene una capacidad de carga máxima, y su objetivo es optimizar la asignación de envíos de manera que se maximice la eficiencia de entrega, respetando la capacidad del vehículo.

  La clase Vehicle representa un vehículo que se utilizará para realizar las rutas de envíos. Tiene atributos como un identificador (id), capacidad de carga (payload), carga actual (current_payload), y una lista de envíos (shipments) que están asignados al vehículo.

  Los métodos de la clase incluyen funciones para obtener y establecer el identificador, la capacidad de carga, la carga actual, y la lista de envíos. También hay un método para llenar el vehículo con la mejor combinación de envíos (fill_vehicle), y métodos de comparación para comparar vehículos según su capacidad de carga.

  La clase Vehicle tiene métodos para serializar y deserializar el objeto, así como una representación en forma de cadena.

Tomando todas las clases anteriores y sus relaciones nos quedaría así el diagrama.


Vehicle
- id: str
- payload: int
- current_payload: int
- shipments: list[Shipment]
+get_id() : : str
+get_payload() : : int
+get_shipments() : : list[Shipment]
+get_current_payload() : : int
+set_id(id: str) : : None
+set_payload(payload: int) : : None
+set_current_payload(current_payload: int) : : None
+set_shipments(shipments: list[Shipment]) : : None
+fill_vehicle(shipments: list[Shipment]) : : None
+__eq__(other: object) : : bool
+__ne__(other: object) : : bool
+__lt__(other: object) : : bool
+__le__(other: object) : : bool
+__gt__(other: object) : : bool
+__ge__(other: object) : : bool
+serialize() : : str
+deserialize(vehicle_json: str) : : None
+__str__() : : str
Route
- shipments: list[Shipment]
+get_shipments() : : list[Shipment]
+set_shipments(shipments: list[Shipment]) : : None
+__str__() : : str
shipments
shipments
Shipment
- client: Client
- product: Product
+get_client() : : Client
+get_product() : : Product
+set_client(client: Client) : : None
+set_product(product: Product) : : None
+serialize() : : str
+deserialize(shipment_dict: dict) : : None
+__str__() : : str
client
product
Client
- _name: Name
- _address: Coordinate
+__init__(name: Name, address: Coordinate)
+get_name() : : Name
+get_address() : : Coordinate
+set_name(name: Name) : : void
+set_address(address: Coordinate) : : void
+__eq__(other: object) : : bool
+__ne__(other: object) : : bool
+__lt__(other: object) : : bool
+__le__(other: object) : : bool
+__gt__(other: object) : : bool
+__ge__(other: object) : : bool
+serialize() : : string
+deserialize(client_json: string) : : void
+__str__() : : string
Product
- _name: string
- _weight: int
+Product(name: string = null, weight: int = null)
+get_name() : : string
+get_weight() : : int
+set_name(name: string) : : void
+set_weight(weight: int) : : void
+__eq__(other: object) : : bool
+__ne__(other: object) : : bool
+__lt__(other: object) : : bool
+__le__(other: object) : : bool
+__gt__(other: object) : : bool
+__ge__(other: object) : : bool
+serialize() : : string
+deserialize(product_json: string) : : void
+__str__() : : string
name
address
Name
- _first: string
- _last: string
+__init__(first: string, last: string) : : void
+get_first() : : string
+get_last() : : string
+set_first(first: string) : : void
+set_last(last: string) : : void
+__eq__(other: object) : : bool
+__ne__(other: object) : : bool
+__lt__(other: object) : : bool
+__le__(other: object) : : bool
+__gt__(other: object) : : bool
+__ge__(other: object) : : bool
+serialize() : : string
+deserialize(name_json: string) : : void
+__str__() : : string
Coordinate
- _x: float
- _y: float
+__init__(x: float, y: float) : : void
+get_x() : : float
+get_y() : : float
+set_x(x: float) : : void
+set_y(y: float) : : void
+get_distance() : : int
+get_distance_to(other: object) : : int
+get_point() : : tuple
+__eq__(other: object) : : bool
+__ne__(other: object) : : bool
+__lt__(other: object) : : bool
+__le__(other: object) : : bool
+__gt__(other: object) : : bool
+__ge__(other: object) : : bool
+serialize() : : string
+deserialize(coordinate_json: string) : : void
+__str__() : : string