Student Tracking System (20 pts)

You are tasked with implementing a simple **Student** class that automatically keeps track of all currently active students in the system.

`Student` Class

This class represents a student and maintains a record of all **active student IDs**.

Class Attribute

- `all_students` — a **set** shared among all instances of the class.
  It stores the `student_id` of every student currently existing in memory.

Instance Attributes

- `name` — the student's full name (string)
- `student_id` — a unique ID for the student (string or integer)

Methods

- `__init__(self, name, student_id)` **(10 pts)**
  Initializes a new student with their name and ID.
  When a new `Student` object is created, its `student_id` should be added to the shared `Student.all_students` set.
- `__del__(self)` **(10 pts)**
  This method should remove the student's `student_id` from the shared `Student.all_students` set.

Example

```
s1 = Student("Alice", "S001")
s2 = Student("Bob", "S002")

print(Student.all_students)
# Expected output: {'S001', 'S002'}

del s1
print(Student.all_students)
# Expected output: {'S002'}
```

Hints

- Use a **set** to store all student IDs to avoid duplicates.
- To remove `student_id` from `all_students`, you can use:

```
all_students.discard(student_id)
```

DMV Vehicle Management System (30 pts)

You are tasked with building a **DMV (Department of Motor Vehicles) management system** that keeps records of different types of vehicles and their owners.

The system will include:

- A **base class** `Vehicle` defining shared attributes and methods.
- Two **subclasses** `Car` and `Truck` extending the base functionality.
- A **DMVRegistry** class to manage all registered vehicles.

`Vehicle` Class (Base Class) (10 pts)

This class represents a generic vehicle registered in the DMV system.

Attributes:

- `vin` — Vehicle Identification Number (string, unique for each vehicle)
- `owner` — name of the vehicle owner (string)
- `type` — string representing the vehicle type (e.g., `"Car"`, `"Truck"`)

Methods:

- `__init__(self, vin, owner, type)`
  Initializes the vehicle's VIN, owner, and vehicle type.
- `def transfer_ownership(self, new_owner):`
  Updates the vehicle's owner to `new_owner`.
- `def get_info(self) -> dict:`
  This method must be **overridden** by subclasses (`Car` and `Truck`) to return a dictionary containing the vehicle's information.
  The base class implementation should **raise an error** to indicate that `Vehicle` does not provide a concrete implementation:

```
        raise NotImplementedError("Subclasses must implement the get_info() method.")
```

`Car` Class (Subclass of `Vehicle`) (5 pts)

Represents a passenger car.

Additional Attribute:

- `num_doors` — number of doors on the car (integer)

Methods:

- `__init__(self, vin, owner, num_doors)`
  Calls the base class initializer using `super()` with `type="Car"` , and adds the `num_doors` attribute.
- `def get_info(self) -> dict:`
  Overrides the base class method to include car-specific details:

```
  {
    "vin": <vin>,
    "owner": <owner>,
    "type": "Car",
    "num_doors": <num_doors>
  }
```

`Truck` Class (Subclass of `Vehicle`) (5 pts)

Represents a commercial or delivery truck.

Additional Attribute:

- `max_load_weight` — maximum cargo capacity in kilograms (float)

Methods:

- `__init__(self, vin, owner, max_load_weight)`
  Calls the base class initializer using `super()` with `type="Truck"` , and adds the `max_load_weight` attribute.
- `def get_info(self) -> dict:`
  Overrides the base class method to include truck-specific details:

```
{
  "vin": <vin>,
  "owner": <owner>,
  "type": "Truck",
  "max_load_weight": <max_load_weight>
}
```

`DMVRegistry` Class (10 pts)

Manages all vehicles registered in the DMV system.

Attributes:

- `vehicles` — a dictionary mapping VINs to `Vehicle` objects.

Methods:

- `def register_vehicle(self, vehicle: Vehicle):`
  Adds a vehicle to the registry.
  If a vehicle with the same VIN already exists, raise:

  ```
  raise ValueError("Vehicle with this VIN is already registered.")
  ```

- `def remove_vehicle(self, vin):`
  Removes a vehicle by its VIN.
  If the VIN does not exist, raise:

  ```
  raise ValueError("Vehicle not found in registry.")
  ```

- `def list_all(self):`
  Prints information for all registered vehicles using each object’s `get_info()` method. One line for one information dictionary.

Example Usage

```
car1 = Car("VIN123", "Alice", 4)
truck1 = Truck("VIN789", "Bob", 5000)

dmv = DMVRegistry()
dmv.register_vehicle(car1)
dmv.register_vehicle(truck1)

car1.transfer_ownership("Charlie")

dmv.list_all()
```

Expected Output:

```
{'vin': 'VIN123', 'owner': 'Charlie', 'type': 'Car', 'num_doors': 4}
{'vin': 'VIN789', 'owner': 'Bob', 'type': 'Truck', 'max_load_weight': 5000}
```