Mini-Project Test: CourierLite

Goal

Build a tiny campus-delivery engine that exercises everything covered so far:

- Collections & data structures (lists, tuples, sets, dicts)
- Object collections (lists/dicts of objects)
- Iterators & generators
- Multiple inheritance/mixins (Python)
- Exceptions & structured logging
- Simple CLI workflow

Use **Python 3.10+**, standard library only.

Provided/Expected Data (put in `data/`)

- `hubs.csv` → `hub_id,hub_name,campus`
- `parcels.csv` → `parcel_id,recipient,priority,hub_id,destination,weight_kg`
    - `priority ∈ {EXPRESS, NORMAL}` (case-insensitive in file; normalize to uppercase)
- `riders.csv` → `rider_id,name,max_load_kg,home_hub_id`

You may add rows, but keep the headers.

Repository Layout (your submission)

```
courierlite/
  models.py
  engine.py
  cli.py
  logging.conf
  README.md
  tests/
data/
  hubs.csv
  parcels.csv
  riders.csv
(optional) pickups.csv
```

Part A — Domain Model & Repos (25 pts)

A1. Mixins & Models (`models.py`)

Create concise, typed, documented classes:

- `class Identifiable:` mixin with `get_id() -> str`
- `class Printable:` mixin with `to_row() -> tuple`
- `class Hub(Identifiable, Printable):` fields: `hub_id, hub_name, campus`
- `class Parcel(Identifiable, Printable):` fields: `parcel_id, recipient, priority (EXPRESS|NORMAL), hub_id, destination, weight_kg: float`
- `class Rider(Identifiable, Printable):` fields: `rider_id, name, max_load_kg: float, home_hub_id`

Implement `__repr__` and lightweight property/getter usage.

A2. In-memory repositories (`engine.py`)

- `HubRepo`, `ParcelRepo`, `RiderRepo` backed by **dict** keyed by id.
- Methods: `add(obj)`, `get(id)`, `all() -> list`, `exists(id) -> bool`, `ids() -> set`.

A3. CSV loading with validation (`engine.py`)

- `load_hubs(path)`, `load_parcels(path)`, `load_riders(path)`
- Normalize priority to uppercase, cast weights/loads to float.
- On a malformed row: **log WARNING** and **skip** (do not crash).

Part B — Assignment & Capacity (30 pts)

B1. Assign parcels (`engine.py`)

Implement:

```
assign_parcels(hubs: HubRepo, riders: RiderRepo, parcels:
ParcelRepo)
-> tuple[dict[str, list[Parcel]], set[str]]
```

Rules:

- A rider can only take parcels from their **home_hub_id**.
- Never exceed `max_load_kg`.
- Assign **EXPRESS** before **NORMAL** (preserve CSV order within each group).
- If multiple riders share a hub, use **round-robin by rider_id (ascending)**; still respect capacity.

Returns:

- `assignments`: dict of `rider_id -> list[Parcel]`
- `unassigned`: set of `parcel_id` that couldn't be assigned

Part C — Iterators & Generators (20 pts)

C1. Custom iterator

`RiderLoadIterator(rider: Rider, parcels: list[Parcel])`
Yields `(parcel_id, weight_kg, cumulative_kg)` in order.

C2. Generator pipelines

- `express_then_normal(parcels_iter)` → yields all EXPRESS, then NORMAL.
- `heavy_first(threshold_kg)` → closure returning a generator that yields parcels ≥ threshold, then the rest.

Demonstrate chaining in the CLI as a "packing preview".

Part D — Exceptions & Logging (15 pts)

D1. Custom exceptions

- `DataFormatError`, `DomainRuleError` (raise where appropriate).

D2. Structured logging

- `logging.conf` → file logs to `logs/courierlite.log`
- Use levels: `INFO` (major steps), `WARNING` (skipped rows/unknown pickups), `ERROR` (fatal).

Part E — CLI (5 pts)

`cli.py` should support:

```
python -m courierlite.cli \
  --hubs data/hubs.csv \
  --parcels data/parcels.csv \
  --riders data/riders.csv \
  --assign \
  --threshold 2.0 \
  [--preview] [--rider R11] [--pickups data/pickups.csv]
```

Output:

- Table per rider: `rider_id | total_load | #parcels | first_3_destinations`
- Sorted `Unassigned:` list of parcel_ids
- If `--preview`, show generator ordering for the specified rider (or for each rider if none specified).

Part F — Tests (5 pts)

Under `tests/` write minimal `unittest` (or `pytest`) covering:

- CSV parsing (malformed row is warned & skipped)
- Assignment respects capacity, hub restriction, EXPRESS priority, round-robin
- Iterator cumulative totals

Part G — Personalization / Anti-Copy Customization (Bonus up to 10 pts)

Each student must add **their own three pickup types** that inherit from a base class and subtly influence ordering. **This section makes your work unique.**

G1. Base + subclasses (`models.py`)

- `class PickupPoint(Identifiable, Printable):`
    - Fields: `pickup_id: str`, `hub_id: str`, `label: str`, `base_priority_bias: int = 0`
    - `to_row()` returns `(pickup_id, label, hub_id, base_priority_bias)`
- Create **three** subclasses with **your own names** (not the examples below), e.g.:
    - `KioskAlpha(PickupPoint)` → `base_priority_bias = +1`
    - `LockerBeta(PickupPoint)` → `base_priority_bias = 0`
    - `DeskGamma(PickupPoint)` → `base_priority_bias = -1`

You must choose different names than your classmates.

G2. Integrate into assignment (`engine.py`)

- Optional CSV `pickups.csv`: `pickup_id,label,hub_id,base_priority_bias`
- If file missing, instantiate your three pickups in code and attach them to existing hubs.
- If a parcel's `destination` contains a token `PICKUP:<pickup_id>`, then when comparing **same-priority** parcels at assignment time, prefer the one whose pickup has **higher `base_priority_bias`**.
- If an unknown `pickup_id` appears → **log WARNING** and ignore bias.

G3. README personalization (**`README.md`**)

Include:

- Your three pickup names, the idea behind them, and how bias affects ordering.
- Your **Student ID** and a checksum:
  `checksum = sum(ord(c) for c in student_id) % 97`
- Use this checksum to seed a tiny randomizer that breaks ties between **NORMAL** parcels **only** (document 1–2 lines of code you used).

This ensures outputs differ across students.

Constraints & Quality Bar

- Python 3.10+, standard library only
- Clear function/method docstrings and type hints
- No globals for core state (use repos/instances)
- Collections returned to the CLI must be deterministic given input and seed
- Handle edge cases gracefully (empty files; no riders on a hub; all parcels overweight; etc.)

How to Run (example)

```
python -m courierlite.cli \
  --hubs data/hubs.csv \
  --parcels data/parcels.csv \
  --riders data/riders.csv \
  --assign --threshold 2.5 --preview --rider R02
```

Marking Rubric (100 pts + bonus)

- **A. Models & repos** — 25
- **B. Assignment logic** — 30
- **C. Iterators & generators** — 20
- **D. Exceptions & logging** — 15
- **E. CLI** — 5
- **F. Tests** — 5
- **G. Personalization (Bonus)** — up to **+10**

Submission

- Zip your `courierlite/` and `data/` folders (and optional `pickups.csv`) as: `<StudentID>_CourierLite.zip`
-
- Include a brief `README.md` (≤ 1 page) with:
    - How to run
    - What you tested
    - Your three pickup types + checksum note

Academic Integrity

Your design, pickup names, biases, and seeded tie-break must be **original**. We will check structure, logs, and seeds for similarity. Collaboration on ideas is fine; code must be your own.

(Optional) Starter Hints

- Keep CSV order by reading rows to a **list**, then sort **within** priority groups only when needed.
- For round-robin, maintain a `hub_id -> index` pointer cycling over eligible riders.
- Write small helpers (e.g., `group_by_hub(riders)`) to keep `assign_parcels` readable.

Good luck, build something tidy, readable, and yours!