https://github.com/AldiyarMaget/Asik2Factory

Report — Evaluation of the written report (clarity, structure, completeness)

Summary:
The report presents a clear design intent: use Factory and Abstract Factory patterns to create asteroids whose speed and radius scale with game level. It includes a working code example, sensible package organization, and emphasizes immutability and separation of concerns.

Clarity:
Class and method names are descriptive (Level, Asteroid, LevelAsteroidFactory, AsteroidFamilyFactory), which makes the code's intent and data flow easy to understand. The public API is minimal and consistent: factories expose createX() methods, products are immutable with getters. The report's prose is straightforward and generally free of ambiguities.

Structure:
The document follows a logical progression: problem statement → design choices → implementation → usage example. The proposed package layout (model, factory, family, app) aligns with common Java conventions and aids maintainability. Code samples are split into focused classes, respecting single responsibility.

Completeness:
The report covers core concerns: immutability, constructor validation, separation of responsibilities, and extensibility for adding new asteroid kinds. It demonstrates how the Abstract Factory produces a family of related asteroids. However, the report lacks practical unit tests, a short UML or diagram to visualize relationships, and explicit notes about thread-safety and configuration of base constants.

Code Quality / Clean Code Assessment:
Strengths:

- Descriptive naming and small focused methods (computeSpeed, computeRadius).
- Immutable product objects (final fields, no setters).
- Constructor validation for guard clauses.
- Delegation: family factory delegates to concrete factories.

Minor improvements:

- Document magic constants (e.g., BASE_SPEED, BASE_RADIUS) and consider making them configurable.
- Clarify naming to avoid potential confusion between class Level and its internal field name.
- Add brief JavaDoc to public classes and methods to help external readers.

Edge cases / Risks:

- Very large level values may cause numeric overflow for computed radius/speed; handling or limits should be noted.
- If Level were mutable, factories holding references could observe external changes — currently Level is immutable which avoids this issue.

Conclusion:
Overall the report effectively explains the design and provides a robust, clean implementation of Factory and Abstract Factory patterns. To raise the report from good to excellent, add unit tests, a simple UML diagram, and short documentation about configuration, threading expectations, and edge-case handling.