STUDI KASUS PBO 1

Dosen Pengampu : Yutika Amelia Effendi, Ph.D.
Mata Kuliah : SIR201 Pemrograman Berorientasi Objek
Prodi : Teknik Robotika dan Kecerdasan Buatan
Kelas : RK-A1 dan RK-A2
Topik : Studi Kasus PBO

Studi Kasus 1 : Sistem Perpustakaan dengan Aturan Pinjam Multi-Tipe

Buatlah sebuah aplikasi perpustakaan berorientasi objek yang mendukung berbagai tipe item (Book, Magazine, DVD), sistem member dengan level, aturan pinjam berbeda per tipe, dan laporan keuangan. Aplikasi harus menggunakan pewarisan, *overriding, overloading*, enkapsulasi, dan polimorfisme.

Spesifikasi code wajib

1. Hirarki kelas
    - `Item` (abstract/base)
        - atribut `id` (string), `title` (string), `available` (bool)
        - konstruktor, getter/setter (enkapsulasi)
        - `virtual int maxLoanDays() const = 0;`
        - `virtual double computeFine(int daysLate) const = 0;`
        - `virtual std::unique_ptr<Item> clone() const = 0;`
    - `Book : public Item`
        - atribut `author`, `isRare` (bool)
        - `maxLoanDays()` default 14; `isRare` ➔ 3 hari
        - `computeFine()` = 1000 * daysLate (book); override.
    - `Magazine : public Item`
        - `maxLoanDays()` = 7; `computeFine()` = 500 * daysLate
    - `DVD : public Item`
        - `maxLoanDays()` = 3; `computeFine()` = 2000 * daysLate
        - overload method `computeFine()` ➔ sediakan overload yang menerima `(int daysLate, bool isDamaged)` sehingga biaya rusak ditambahkan.
2. Kelas Member
    - `Member` (base)
        - atribut `id`, `name`, `level` (int), `freeLoanCredits` (int)

- method `borrow(Item &it, int daysRequested)` ➔ harus mengecek `maxLoanDays()`, update `available`, kurangi credits jika berlaku.
- overloading `borrow` juga untuk `borrow(Item &it, int daysRequested, bool useCredit)` ➔ gunakan credits jika member ingin.
- method `returnItem(Item &it, int actualDays)` ➔ menghitung denda via `Item::computeFine` (polimorfik), update `available`.

  - Level rules:
    - Level 0 : no free credits
    - Level 1 : 1 free credit per 5 successful loans
    - Level 2 : 3 credits per 10 loans
    - Mahasiswa harus implementasikan mekanisme kenaikan level berdasarkan jumlah pinjam historis

3. Pengelolaan koleksi dan polimorfisme
   - `std::vector<std::unique_ptr<Item>> catalogue;`
   - Fungsi pencarian berdasarkan `id`, `title`.
   - Saat memanggil `computeFine` gunakan pointer/reference ke `Item` (polimorfisme).
4. Fitur tambahan wajib
   - Operator overloading untuk membandingkan dua `Item` (mis. < berdasarkan `id`) agar dapat sorting katalog.
   - Implementasikan `clone()` untuk mendukung *deep copy* koleksi.
   - Laporan bulanan: jumlah peminjaman per tipe, total denda, total free credits terpakai.
   - Validasi input sederhana (IDs unik, daysRequested >0).
5. I/O dan antarmuka
   - Bisa CLI menu sederhana: add item, register member, borrow, return, report.
   - Wajib ada file `testcases.txt` yang mendemokan minimal 8 alur: pendaftaran, pinjam/return, penggunaan credit, denda, upgrade level.

Tugas kelompok (pembagian peran (*opsional, tergantung kelompok*))
Note : Setiap anggota kelompok harus mempunyai tugas

- Anggota 1 ➔ Arsitek dan core Item classes (Item, Book, Magazine, DVD, clone, operator).
- Anggota 2 ➔ Member management dan level rules (Member class, borrow/return overload).
- Anggota 3 ➔ Catalogue dan persistence (search, sort, clone koleksi, save/load sederhana).
- Anggota 4 ➔ CLI / orchestrator (menu, input handling, integrasi).

- Anggota 5 ➔ Testing dan dokumentasi (buat `testcases.txt`, README, laporan dan demo).

Hasil Project

- Kode program lengkap
- README (cara compile/run, peran/job setiap anggota)
- `testcases.txt` dan output yang diharapkan
- Laporan project (desain kelas, alasan OOP choices, I/O, testcases min 8 alur) *[format bebas]*
- Video demo (maksimal 15 menit) *[format dan tools bebas]*

Kasus uji minimal

1. Tambah 3 item berbeda (Book, Magazine, DVD); sort katalog.
2. Register member; pinjam Book selama 10 hari (dapat/ditolak sesuai maxLoanDays).
3. Pinjam DVD dan kembalikan terlambat 2 hari; perhitungan denda menggunakan overload `computeFine(daysLate, isDamaged)` vs `computeFine(daysLate)`.
4. Gunakan free credit pada member; cek pengurangan credit.
5. Generate report bulanan: total denda, peminjaman per tipe.

Studi Kasus 2: Simulator Fleet Robot Polymorfik

Buat sebuah mini-simulator C++ untuk fleet kendaraan robotik (GroundRobot, AerialDrone, Amphibious) yang memodelkan pergerakan, sensor, dan aksi khusus. Sistem harus menerapkan inheritance, overriding, overload, polimorfisme, serta mekanisme *command* sederhana.

Spesifikasi code wajib

1. Hirarki kelas
    - `Vehicle` (abstract/base)
        - atribut protected: `id`, `position` (struct {double x,y,z}), `battery` (double)
        - `virtual void move(double dx, double dy, double dz) = 0;`
        - `virtual void status() const;`
        - `virtual std::unique_ptr<Vehicle> clone() const = 0;`
        - `virtual ~Vehicle()` virtual destructor.
        - method `bool consumeBattery(double amount)` ➔ mengurangi battery, return false jika habis.

- `GroundRobot : public Vehicle`
  - `move(double dx, double dy, double dz)` — `dz` diabaikan (ground), pergerakan terbatas max slope, gunakan pemeriksaan parameter (override).
  - overload `move(double heading, double distance)` ➔ hitung dx/dy dari heading dan distance (degrees/radians).
- `AerialDrone : public Vehicle`
  - `move(double dx, double dy, double dz)` ➔ pergerakan 3D, battery consumption berbeda.
  - overload `move(const Position &target)` ➔ bergerak ke koordinat absolut.
  - method `override status()` menampilkan altitude, battery, posisi.
- `Amphibious : public GroundRobot, public AerialDrone`
  - harus menggabungkan kemampuan bergerak di darat dan udara/air (boleh pilih composition: punya `GroundModule` dan `AerialModule`).

2. *Command dan polymorphism*
   - Simpan semua kendaraan di `std::vector<std::unique_ptr<Vehicle>> fleet;`
   - Implementasikan fungsi `executeMove(Vehicle &v, /*various signatures*/)` yang memanggil overload `move` sesuai input.
   - Gunakan `dynamic_cast` aman untuk memeriksa tipe saat perlu (misal: hanya `AerialDrone` bisa `move` in 3D secara penuh).
3. *Sensor dan collision handling*
   - Simulasikan sensor sederhana: `bool obstacleAhead(double range)` virtual function (base default false); override di `GroundRobot` dengan hardcoded obstacles list (vector positions).
   - Jika obstacleDetected ➔ `move()` harus mengurangi langkah atau menolak perintah.
4. *Overloading dan overriding* (wajib)
   - Pastikan setidaknya satu metode memiliki overloading (misal: `move` beberapa signature) dan satu metode overriding (misal: `status`, `obstacleAhead`).
5. *Scenario dan test*
   - Siapkan skenario misi: 3 kendaraan (1 GroundRobot, 1 AerialDrone, 1 Amphibious) menempuh rute berurutan (waypoints).
   - Simulasikan battery drain, obstacle detection, dan fallback behavior (GroundRobot berhenti; AerialDrone naik ketinggian; Amphibious berpindah ke mode lain).

Tugas kelompok (pembagian peran (*opsional, tergantung kelompok*))
Note : Setiap anggota kelompok harus mempunyai tugas

- Anggota 1 ➔ Desain dan Vehicle base class (Position struct, clone())
- Anggota 2 ➔ GroundRobot dan obstacles (override move, obstacleAhead)
- Anggota 3 ➔ AerialDrone dan status reporting (overloaded move, status)
- Anggota 4 ➔ Amphibious (integration) dan fleet manager (vector, executeMove)
- Anggota 5 ➔ Testing, scenario scripts dan dokumentasi (test cases, README, demo run)

Hasil Project

- Kode program lengkap
- README (cara compile/run, peran/job setiap anggota) + Run Script (*sample commands*)
- `scenarios.txt` berisi waypoint sequences dan expected logs
- Laporan project (desain kelas, alasan OOP choices, I/O, testcases) *[format bebas]*
- Video demo (maksimal 15 menit) *[format dan tools bebas]*

Kasus uji minimal

1. GroundRobot: coba `move(1.0, 0.0, 0.0)` di area tanpa obstacle ➔ posisi berubah sesuai, battery turun.
2. GroundRobot: `move` menuju obstacle dalam jarak range ➔ perintah ditolak / langkah dikurangi.
3. AerialDrone: `move(targetPosition)` ➔ bergerak ke target, battery drain sesuai jarak.
4. Fleet: jalankan misi 3 waypoints, log tiap kendaraan: posisi awal, setiap move, battery, dan jika gagal.

Selamat Mengerjakan