Laborator 10 – Vectori de obiecte neomogene

Tema: Racing Garage 🚗

Autor: ing. Eduard Donea

Poveste

Într-un garaj modern dedicat pasionaților de automobile, o colecție variată de vehicule este pregătită pentru sesiuni de test-drive. Fiecare model – fie că este o Dacia ușoară, un Ford solid sau un BMW sportiv – are propriile caracteristici tehnice și stil de conducere. În cadrul laboratorului, studenții vor construi un mic sistem OOP care simulează un garaj, în care vehiculele sunt testate polimorfic printr-o sesiune unică de test-drive.

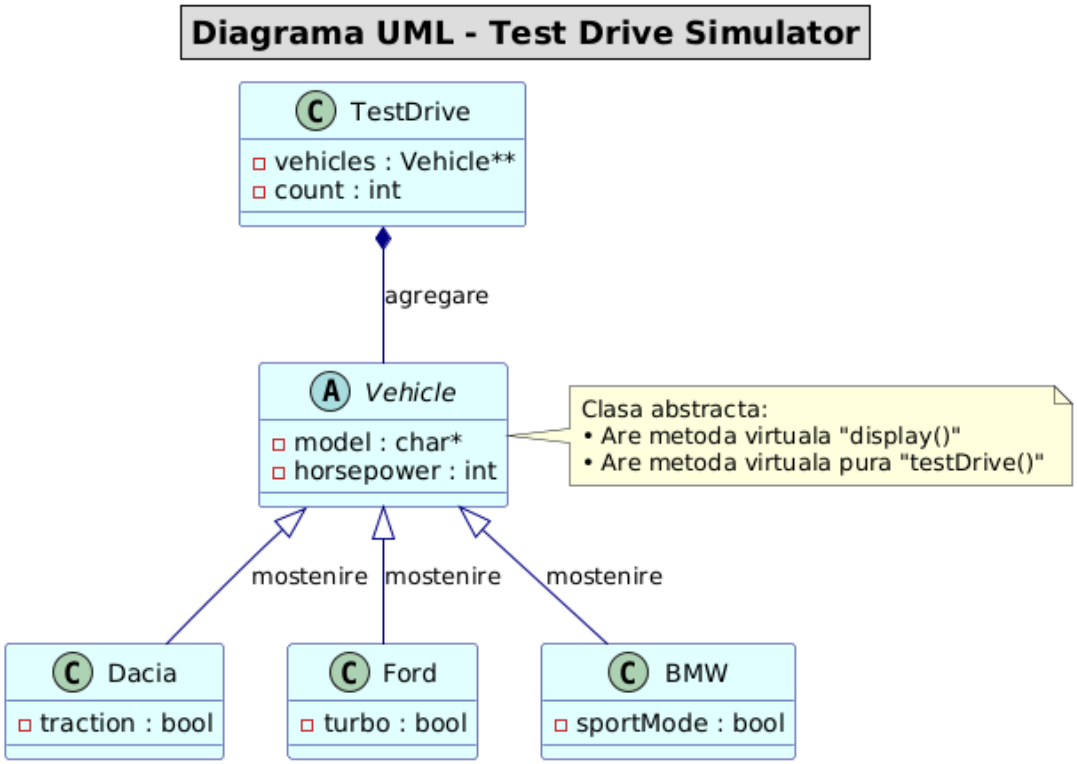
Scopul laboratorului

- înțelegerea conceptului de **clasă abstractă** și metode **virtuale**;
- folosirea polimorfismului prin pointeri la clasa de bază;
- diferențierea relațiilor `is-a` și `has-a`;
- definirea unui operator suprascris (`operator<<`);
- organizarea corectă a unui proiect C++ modular.

Structura proiectului

- **Vehicle** – clasa abstractă (model, horsepower, metode virtuale);
- **Dacia**, **Ford**, **BMW** – exemple de clase derivate care suprascriu test-drive-ul;
- **TestDrive** – clasa care gestionează un vector de pointeri la Vehicle și simulează rularea;
- **main.cpp** – programul principal unde sunt create vehiculele și pornește test-drive-ul.

Figura 1 – Diagrama UML a sistemului Racing Garage



Clasa abstractă: Vehicle

Clasa `Vehicle` reprezintă superclasa tuturor vehiculelor din garaj. Ea definește atributele comune – modelul și puterea motorului – și un comportament polimorfic. Metoda `testDrive()` este declarată virtual pură, obligând fiecare marcă de mașină să-și definească propriul comportament în timpul test-drive-ului.
De asemenea, metoda virtuală `print()` va permite suprascrierea operatorului ¡¡, astfel încât orice vehicul să poată fi afișat ușor.

Clase derivate: Dacia, Ford, BMW

Clasa `Dacia` oferă un model accesibil, cu putere redusă și stil de condus stabil. Implementarea metodei `testDrive()` va reflecta caracteristicile unei mașini economice.

`Ford` reprezintă un vehicul fiabil, cu putere medie și un comportament mai agresiv față de Dacia. Test-drive-ul va simula un stil de condus echilibrat, dar ferm

Clasa `BMW` ilustrează o mașină premium, cu performanțe ridicate. Test-drive-ul trebuie să reflecte un stil sportiv, accelerare rapidă și manevrabilitate superioară.

OBSERVAȚIE: Sunt niște exemple, voi puteți implementa ce modele de mașini doriți.

Clasa `TestDrive`

Clasa `TestDrive` gestionează o colecție de pointeri la obiecte de tip `Vehicle`. Ea nu știe ce tip concret de vehicul găzduiește (Dacia, Ford, BMW etc.), ceea ce permite demonstrarea polimorfismului la runtime.
Metoda `startSession()` va parcurge vectorul de vehicule și va apela polimorfic `testDrive()` pentru fiecare, simulând o sesiune reală de testare auto.

Fișierul `main.cpp`

```cpp
int main() {
    // TODO: creati cateva vehicule (Dacia, Ford, BMW)
    // TODO: creati un obiect TestDrive
    // TODO: adaugati vehiculele in vector
    // TODO: apelati sesiunea de test-drive
    // TODO: afisati vehiculele folosind operator<<
    // TODO: eliberati memoria
    return 0;}
```

Cerințe (10p)

1. (3p) Implementați clasa abstractă `Vehicle` (constructori, destructor, deep copy dacă este cazul, metodele virtuale).
2. (3p) Implementați clasele derivate `Dacia`, `Ford`, `BMW` (sau altele) și comportamentul lor în `testDrive()`.
3. (2p) Implementați clasa `TestDrive` și gestionați corect vectorul de pointeri.
4. (1p) Suprascrieți operatorul `<<` pentru afișarea vehiculelor.
5. (1p) Realizați o sesiune completă de test-drive în `main.cpp` folosind polimorfism.

Cerință Bonus (1p) – Ștergerea unui vehicul din garaj

Pentru punctaj suplimentar, implementați în clasa `TestDrive` o metodă:

```
void removeVehicle(const char* modelName);
```

Această metodă trebuie să permită eliminarea în siguranță a unui vehicul din vectorul intern (`Vehicle** vehicles`), pe baza numelui modelului. Funcționalitatea cerută:

- căutarea vehiculului după nume;
- eliberarea memoriei obiectului găsit;
- realocarea vectorului pentru eliminarea poziției curente;
- actualizarea corectă a numărului de vehicule;
- tratarea cazurilor speciale: vector gol, model inexistent, un singur element.

După eliminare, garajul trebuie să rămână într-o stare validă, iar sistemul să permită în continuare rularea unei sesiuni de test drive fără erori.

Compilare și rulare

```
g++ -g *.cpp -o lab10
valgrind ./lab10
```

Bonus GitHub Classroom

Studentii pot obtine pana la **3p bonus** daca incarca toate laboratoarele pana la finalul semestrului, respectiv **1p bonus** daca incarca mai mult de jumatate dintre ele.

Pasii pentru incarcarea laboratorului:

1. Accesati cursul de pe OCW – Programare Orientata pe Obiecte, sectiunea **Resurse** → **Bonus GitHub laborator**.
2. Acolo veti gasi linkul **GitHub Classroom** pentru Laboratorul 10:

   `https://classroom.github.com/a/oCC3Uas3`

3. Dupa acceptare, veti primi propriul repository, ex. `laborator-10-NumePrenume`.
4. Clonati repository-ul local:

```
git clone
↪  git@github.com:Laborator-POO-2025-2026/laborator-10-NumePrenume.git
cd laborator-10-NumePrenume
```

5. Dupa ce ati adaugat fisierele, urmeaza comenzile:

```
git add .
git commit -m "Mesaj de commit"
git push origin main
```

Atenție: laboratoarele copiate între studenți sunt considerate nevalide, iar niciunul dintre cei doi participanți nu va primi bonusul. Repository-urile sunt monitorizate automat de sistemul GitHub Classroom.