Tema 2 Sistem de Cunoștințe Bazat pe Cadre pentru Ierarhia Vehiculelor

Sisteme Semantice

Draghici Andreea-Maria
Inginerie Software
IS 2.1

Tema 2 Sistem de Cunoștințe Bazat pe Cadre pentru Ierarhia Vehiculelor

Contents

Table of figures

Tema 2 Sistem de Cunoștințe Bazat pe Cadre pentru Ierarhia Vehiculelor

1. Introducere

Acest proiect prezintă un sistem de cunoștințe bazat pe cadre, organizat ierarhic, pentru a modela vehiculele și relațiile dintre ele. Structura codului este inspirată de relațiile de **Generalizare/Specializare** și **Agregare**, iar implementarea este realizată în limbajul Java.

2. Scopul proiectului

Scopul acestui proiect este de a dezvolta un sistem flexibil de reprezentare a vehiculelor și a caracteristicilor lor, într-o structură ierarhică pe mai multe nivele. Prin utilizarea de atribute calculabile și a unui model bazat pe cadre, proiectul este capabil să răspundă la interogări detaliate despre atributele vehiculelor.

3. Structura Ierarhică a Proiectului

Sistemul este organizat pe **5 nivele,** fiecare nivel având un rol specific în modelarea comportamentului și caracteristicilor vehiculului:

Nivel 1: Vehicul

- **Tip:** Clasă de bază (abstractă)
- **Descriere:** Clasa fundamentală pentru toate tipurile de vehicule. Definește atributele și metodele comune pentru vehicule.
- **Conține următoarele metode:**
  1. **adaugaProprietate(String numeProprietate, Object valoare):** Adaugă o proprietate în proprietati.
  2. **obtineValoareProprietate(String numeProprietate):** Returnează valoarea pentru o proprietate specificată.
  3. **afiseazaProprietati():** Afișează toate proprietățile și valorile vehiculului.

Nivel 2: VehiculMotorizat

- **Tip:** Subclasă a Vehicul
- **Descriere:** Reprezintă vehiculele care au motor. Conține atribute specifice legate de performanța motorului. (putereMotor, calculVitezaMaxima, calculConsumCombustibil, calculAutonomie).

Nivel 2: VehiculNemotorizat

- **Tip:** Subclasă a Vehicul
- **Descriere:** Reprezintă vehiculele fără motor. Include biciclete și trotinete.

Nivel 3: VehiculPersonal

- **Tip:** Subclasă a VehiculMotorizat
- **Descriere:** Dedicată vehiculelor utilizate în scop personal, cum ar fi automobilele.
- **Conține:** un atribut ce specifică utilizarea vehiculului, setat la –personal–.

Nivel 3: VehiculDeTransport

- **Tip:** Subclasă a VehiculMotorizat
- **Descriere:** Dedicată vehiculelor utilizate pentru transportul de mărfuri, cum ar fi camioanele.
- **Conține:** Un atribut care specifică utilizarea vehiculului, setat la –transport–.

Nivel 4: Automobil

- **Tip:** Subclasă a VehiculPersonal
- **Descriere:** Reprezintă automobilele, având caracteristici specifice precum tipul combustibilului și capacitatea rezervorului.
- **Instanță specifică:** MasinaFamilie - o instanță particulară a clasei Automobil, care poate avea atribute definite pentru o mașină de familie.

Nivel 4: Camion

- **Tip:** Subclasă a VehiculDeTransport
- **Descriere:** Reprezintă camioanele, care sunt utilizate pentru transportul de marfă.
- **Conține:** Atribut pentru capacitateaIncarcare ce reprecina capacitatea de încărcare a camionului o instanță a clasei RezervorCombustibil și a clasei Motor.
- **Instanță specifică:** CamionMarfa - o instanță particulară a clasei Camion, care poate avea atribute specifice pentru un camion de marfă.

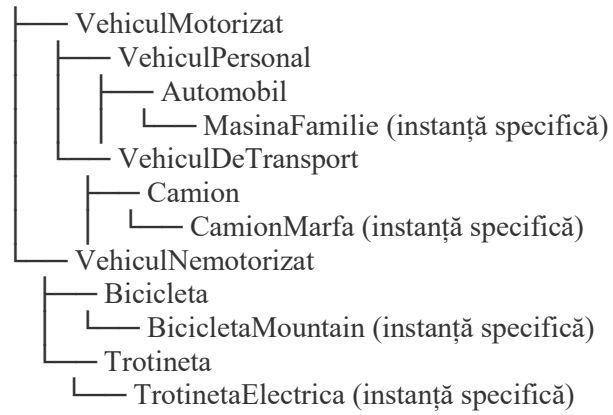
Nivel 3: Bicicleta

- **Tip:** Subclasă a VehiculNemotorizat
- **Descriere:** Reprezintă bicicletele, care sunt vehicule fără motor.
- **Conține:** Un atribut care indică tipul cadrului bicicletei.
- **Instanță specifică:** BicicletaMountain - o instanță particulară a clasei Bicicleta, specializată pentru utilizarea pe teren accidentat.

Nivel 3: Trotineta

- **Tip:** Subclasă a VehiculNemotorizat
- **Descriere:** Reprezintă trotinetelor, care sunt de asemenea vehicule fără motor.
- **Conține:** Un atribut de tipul Boolean care indică daca trotineta este electrică.
- **Instanță specifică:** TrotinetaElectrica - o instanță particulară a clasei Trotineta, care este electrică.

Diagrama UML pentru a reprezenta ierarhia:



4. Implementarea Pattern-ului Bunch

Pentru a permite specificarea flexibilă a atributelor fiecărui vehicul, am implementat **Pattern-ul Bunch**, utilizând o structură de date Map<String, Object> în fiecare clasă. Acest Map permite:

- Stocarea atributelor și a valorilor într-un format flexibil, fără a necesita o definiție fixă a proprietăților pentru fiecare tip de vehicul.

```
abstract class CadruVehicul {
    protected String nume;
    protected Map<String, Object> proprietati;

    public CadruVehicul(String nume, Map<String, Object> proprietatiInitiale) {
        this.nume = nume;
        this.proprietati = new HashMap<>(proprietatiInitiale);
    }

    public CadruVehicul(String nume) {
        this.nume = nume;
        this.proprietati = new HashMap<>();
    }

    public void adaugaProprietate(String numeProprietate, Object valoare) {
        proprietati.put(numeProprietate, valoare);
    }

    public Object obtineValoareProprietate(String numeProprietate) {
        Object valoare = proprietati.getOrDefault(numeProprietate, "undefined");
        if (valoare instanceof CalculatorProprietate) {
            return ((CalculatorProprietate) valoare).calcul();
        }
        return valoare;
    }

    public void afiseazaProprietati() {
        System.out.println("Proprietăți pentru " + nume + ":");
        for (String numeProprietate : proprietati.keySet()) {
            System.out.println(numeProprietate + ": " + obtineValoareProprietate(numeProprietate));
        }
    }
}
```

- Gestionarea atributelor calculabile, prin utilizarea unei interfețe CalculatorProprietate care permite calculul dinamic al unor valori (cum ar fi viteza maximă de exemplu).

```
// Interfata pentru calcularea atributelor dinamice
interface CalculatorProprietate {
    Object calcul();
}
```

Avantajele Pattern-ului Bunch:

1. **Flexibilitate**: Permite definirea atributelor variabile, specifică prototipurilor de vehicule complexe.
2. **Calcul dinamic al atributelor**: Utilizând CalculatorProprietate, proprietățile pot fi calculate doar la nevoie, ceea ce optimizează performanța și memoria.
3. **Reutilizabilitate**: Clasa CadruVehicul și subclasele sale pot fi extinse ușor pentru alte tipuri de vehicule.

5. Descrierea Codului și a Componențelor Principale

5.1 Clase și Relații

Proiectul este organizat într-o structură ierarhică de clase care modelază diferite tipuri de vehicule. Fiecare clasă are propriile sale atribute și metode care reflectă caracteristicile specifice ale vehiculului pe care îl reprezintă.

Componentele Principale:

1. **Clasa de bază CadruVehicul**
2. **Clasa Vehicul**
3. **Subclasa VehiculMotorizat**
4. **Subclasa VehiculNemotorizat**
5. **Subclasa VehiculPersonal**
6. **Subclasa VehiculDeTransport**
7. **Clasa Automobil**
8. **Clasa Camion**
9. **Clasa Bicicleta**
10. **Clasa Trotineta**
11. **Clasa Main** (punctul de intrare al programului)

Fiecare clasă din acest proiect poate fi considerată un cadru care conține atribute ce descriu caracteristicile specifice ale vehiculului.

Componentele de Tip Agregare:

În proiect, agregarea este utilizată pentru a reprezenta relațiile între clasele care descriu diferitele componente ale unui vehicul.

Clasa Motor

- **Descriere:** Clasa Motor reprezintă componenta motorului unui vehicul.
- **Atribute:**
  - putere: Puterea motorului, exprimată în cai putere (CP).
  - tipCombustibil: Tipul de combustibil utilizat (ex: benzină, motorină).
- **Agregare:** Un vehicul motorizat (de exemplu, un automobil sau un camion) va conține un obiect de tip Motor.

Clasa RezervorCombustibil

- **Descriere:** Clasa RezervorCombustibil reprezintă rezervorul de combustibil al vehiculului.
- **Atribute:**
  - capacitate: Capacitatea rezervorului de combustibil, exprimată în litri.
- **Agregare:** Similar cu motorul, un vehicul motorizat va conține un obiect de tip RezervorCombustibil

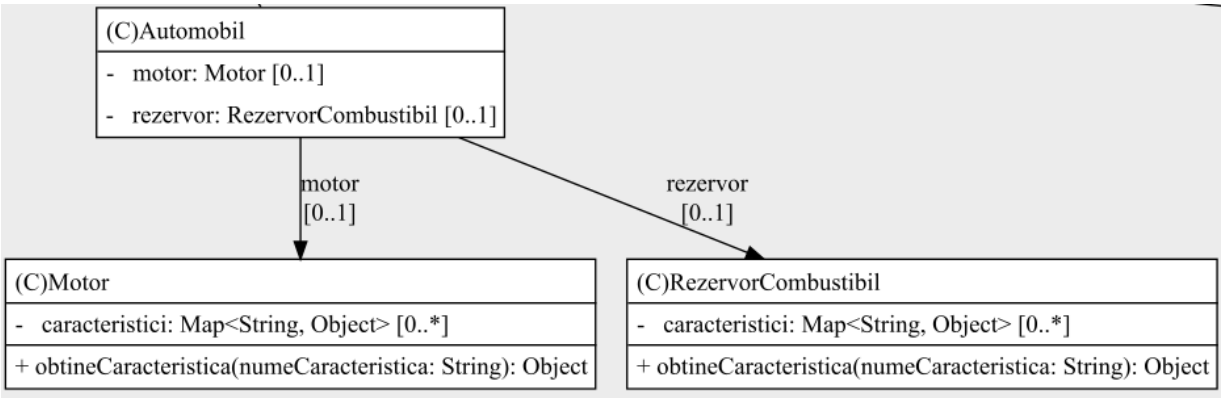


Figure 1 Diagrama UML - Modelul Agregării

Avantajele Utilizării Agregării

1. **Modularitate:** Agregarea permite separarea componentelor vehiculului în clase distincte, facilitând gestionarea și întreținerea codului. De exemplu, modificările aduse clasei Motor nu afectează direct clasa Masina, cu excepția cazului în care sunt necesare actualizări ale interfeței.
2. **Reutilizare:** Componentele, cum ar fi Motor și RezervorCombustibil, pot fi reutilizate în diferite tipuri de vehicule, reducând astfel duplicarea codului și îmbunătățind eficiența.
3. **Claritate:** Structura ierarhică clară, bazată pe agregare, face ca relațiile dintre clase să fie ușor de înțeles.

Relațiile dintre Clase:

- **Moștenire:**
  - VehiculMotorizat moștenește de la Vehicul.
  - VehiculNemotorizat moștenește de la Vehicul.
  - VehiculPersonal moștenește de la VehiculMotorizat.
  - VehiculDeTransport moștenește de la VehiculMotorizat.
  - Automobil moștenește de la VehiculPersonal.
  - Camion moștenește de la VehiculDeTransport.
  - Bicicleta moștenește de la VehiculNemotorizat.
  - Trotineta moștenește de la VehiculNemotorizat.

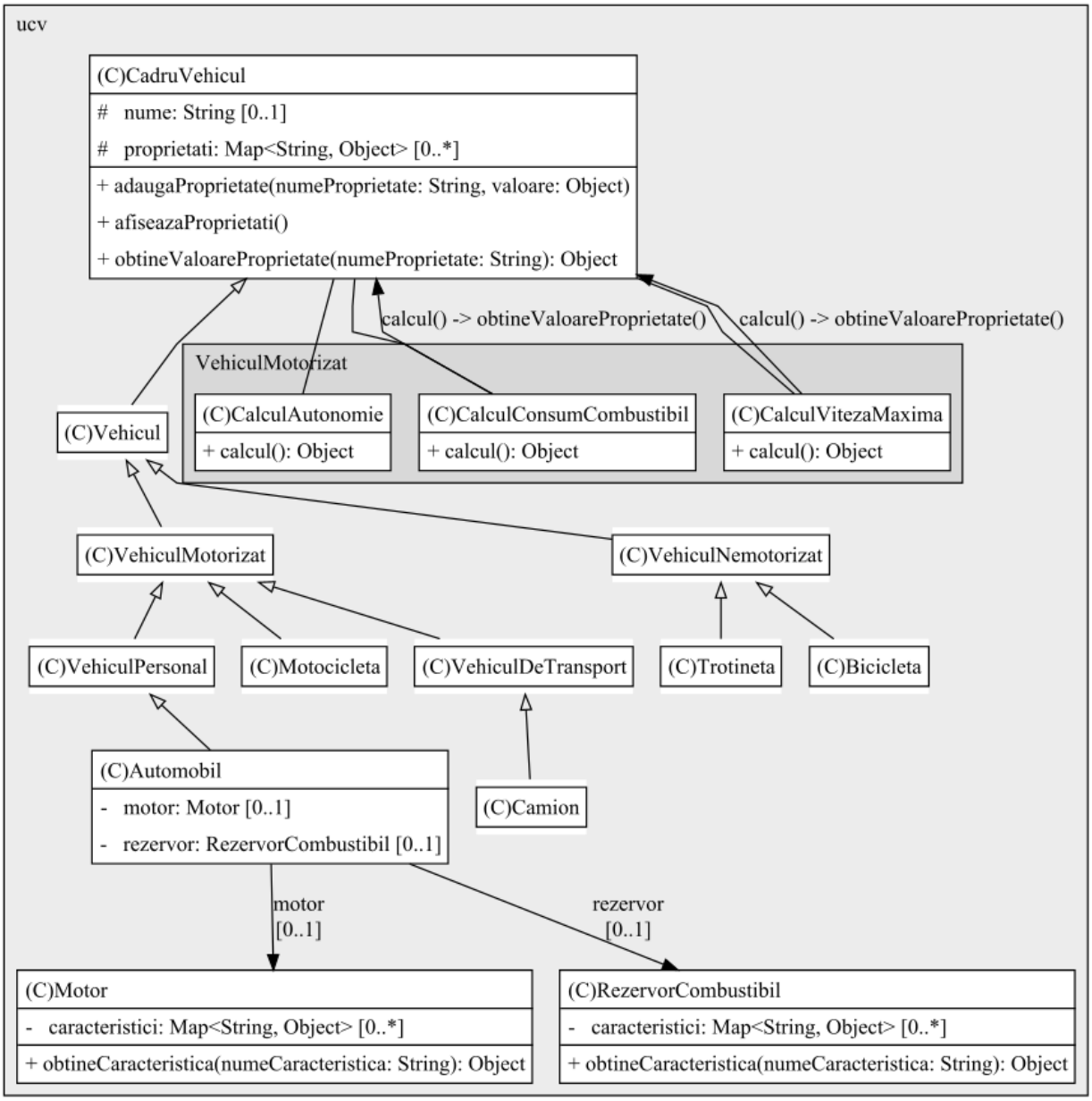


Figure 2 Diagrama UML - Relațiile Sistemului de Modelare a Vehiculelor

5.2 Calculul Atributelor și Interogarea Informațiilor

Procedura: CalculVitezaMaxima

- **Descriere:** Această metodă calculează viteza maximă a vehiculului pe baza puterii motorului. Formula utilizată în acest calcul înmulțește puterea motorului cu un coeficient (1.5 în acest caz).

```
private class CalculVitezaMaxima implements CalculatorProprietate {
    public Object calcul() {
        int putereMotor = (Integer) obtineValoareProprietate("putereMotor");
        return putereMotor * 1.5;
    }
}
```

Procedura: CalculConsumCombustibil

- **Descriere:** Această metodă calculează consumul de combustibil al vehiculului, folosind puterea motorului. Formula aplicată împarte puterea motorului la 10, pentru a obține un rezultat relevant pentru consumul de combustibil.

```
private class CalculConsumCombustibil implements CalculatorProprietate {
    public Object calcul() {
        int putereMotor = (Integer) obtineValoareProprietate("putereMotor");
        return putereMotor / 10.0;
    }
}
```

Procedura: CalculAutonomie

- **Descriere:** Această metodă calculează autonomia vehiculului, adică distanța maximă pe care vehiculul o poate parcurge cu un plin de combustibil. Calculul se bazează pe capacitatea rezervorului și consumul de combustibil.

```
private class CalculAutonomie implements CalculatorProprietate {
    public Object calcul() {
        Object consumCombustibilObj = obtineValoareProprietate("calculConsumCombustibil");
        Object capacitateRezervorObj = obtineValoareProprietate("capacitateRezervor");

        // Verificăm dacă consumCombustibil și capacitateRezervor sunt de tipul corect
        if (consumCombustibilObj instanceof Double && capacitateRezervorObj instanceof
Integer) {
            double consumCombustibil = (Double) consumCombustibilObj;
            int capacitateRezervor = (Integer) capacitateRezervorObj;
            return (capacitateRezervor / consumCombustibil) * 100;
        } else {
            return "undefined"; // dacă valorile nu sunt de tipul corect
        }
    }
}
```

5.2.2 Exemplu de Interogări și Răspunsuri

În clasa Main, putem realiza interogări asupra fiecărei instanțe de vehicul folosind obtineValoareProprietate și afiseazaProprietati, care oferă:

1. **Răspunsuri exacte** pentru atributele disponibile.
2. **Valoarea undefined** pentru atributele nedefinite, conform cerințelor de proiect.

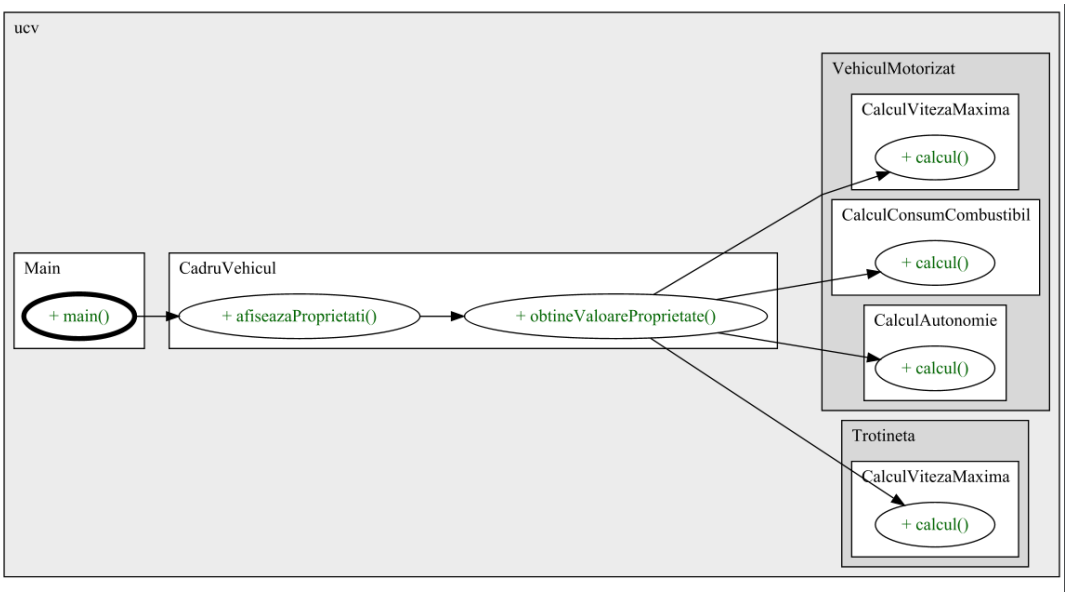


Figure 3 Diagrama UML - Modul în care obținem și afișăm proprietățile

Utilizatorii pot accesa valorile atributelor vehiculului printr-un sistem de interogare bazat pe chei, facilitând obținerea rapidă a informațiilor necesare.
Metoda de afișare a proprietăților permite utilizatorilor să vizualizeze rapid toate informațiile relevante despre un vehicul, oferind o imagine de ansamblu clară asupra caracteristicilor acestuia.

5.2.3 Exemple de ieșire

Pentru o instanță specifică, masinaFamilie, afișarea proprietăților va include:

Detalii Automobil - Masina de Familie:
Proprietăți pentru Vehicul:
calculVitezaMaxima: 375.0
putereMotor: 250
utilizare: personal
calculAutonomie: 240.0
calculConsumCombustibil: 25.0
putere: 250
tipCombustibil: Diesel
capacitateRezervor: 60

Aceasta demonstrează capacitatea sistemului de a calcula atributele și de a returna valoarea calculată.

6. Simulare sistem

```
package ace.ucv;

import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        // Definirea caracteristicilor pentru Motor și Rezervor
        Map<String, Object> proprietatiMotor = new HashMap<>();
        proprietatiMotor.put("putere", 250); // Integer
        proprietatiMotor.put("tipCombustibil", "Diesel");

        Map<String, Object> proprietatiRezervor = new HashMap<>();
        proprietatiRezervor.put("capacitate", 60); // Integer

        // Crearea unei instanțe de Automobil, care este un VehiculPersonal
        Automobil masinaFamilie = new Automobil(proprietatiMotor, proprietatiRezervor);

        // Caracteristicile pentru Camion
        Map<String, Object> proprietatiMotorCamion = new HashMap<>();
        proprietatiMotorCamion.put("putereMotor", 400); // Integer pentru camion
        proprietatiMotorCamion.put("tipCombustibil", "Motorina");

        Map<String, Object> proprietatiRezervorCamion = new HashMap<>();
        proprietatiRezervorCamion.put("capacitateRezervor", 120); // Integer mai mare pentru camion

        // Instanța de Camion
        Camion camionMarfa = new Camion(proprietatiMotorCamion, proprietatiRezervorCamion);

        // Instanța de Bicicleta
        Map<String, Object> proprietatiBicicleta = new HashMap<>();
        proprietatiBicicleta.put("tipCadru", "Carbon");
        Bicicleta bicicletaMountain = new Bicicleta(proprietatiBicicleta);

        // Instanța de Trotineta
        Map<String, Object> proprietatiTrotineta = new HashMap<>();
        proprietatiTrotineta.put("esteElectrica", true);
        Trotineta trotinetaElectrica = new Trotineta(proprietatiTrotineta);

        // Instanța de Motocicleta
        Map<String, Object> proprietatiMotocicleta = new HashMap<>();
        proprietatiMotocicleta.put("putereMotor", 10);
        proprietatiMotocicleta.put("tipCombustibil", "Benzina");
        proprietatiMotocicleta.put("casca", true);

        // Instanța de Motocicleta
        Motocicleta motocicletaSport = new Motocicleta(proprietatiMotocicleta);

        System.out.println("Detalii Automobil - Masina de Familie:");
        masinaFamilie.afiseazaProprietati();

        System.out.println("\nDetalii Bicicleta - Bicicleta Mountain:");
        bicicletaMountain.afiseazaProprietati();

        System.out.println("\nDetalii Trotineta - Trotineta Electrica:");
        trotinetaElectrica.afiseazaProprietati();


        System.out.println("\nDetalii Camion - Camion de Marfa:");
        camionMarfa.afiseazaProprietati();
```

```
        System.out.println("\nDetalii Motocicleta - Motocicleta Sport:");
        motocicletaSport.afiseazaProprietati();

    }
}
```

Rezultate obținute, afișarea proprietăților:

Detalii Automobil - Masina de Familie:
Proprietăți pentru Vehicul:
calculVitezaMaxima: 375.0
putereMotor: 250
utilizare: personal
calculAutonomie: 240.0
calculConsumCombustibil: 25.0
putere: 250
tipCombustibil: Diesel
capacitateRezervor: 60

Detalii Bicicleta - Bicicleta Mountain:
Proprietăți pentru Vehicul:
tipCadru: Carbon

Detalii Trotineta - Trotineta Electrica:
Proprietăți pentru Vehicul:
calculVitezaMaxima: 25
esteElectrica: true

Detalii Camion - Camion de Marfa:
Proprietăți pentru Vehicul:
calculVitezaMaxima: 600.0
putereMotor: 400
utilizare: transport
capacitateIncarcare: 10000
calculAutonomie: 300.0
calculConsumCombustibil: 40.0
tipCombustibil: Motorina
capacitateRezervor: 120

Detalii Motocicleta - Motocicleta Sport:
Proprietăți pentru Vehicul:
calculVitezaMaxima: 15.0
calculAutonomie: undefined
casca: true
calculConsumCombustibil: 1.0
putereMotor: 10
tipCombustibil: Benzina

7. Concluzii

- **Implementare Eficientă a OOP:** Tema a demonstrat aplicarea cu succes a principiilor programării orientate pe obiecte (OOP) prin utilizarea moștenirii, polimorfismului și encapsulării. Structura ierarhică a claselor a permis organizarea logică a vehiculelor, facilitând reutilizarea codului și extinderea ulterioară a funcționalităților.

- **Flexibilitate și Extensibilitate:** Utilizarea unui HashMap pentru gestionarea atributelor vehiculelor a permis o flexibilitate sporită în adăugarea și modificarea proprietăților. Această abordare face ca sistemul să fie ușor de adaptat la noi cerințe și extinderi, fără a necesita modificări semnificative în structura codului.

- **Calculul Atributelor:** Metodele de calcul implementate pentru viteza maximă, consumul de combustibil și autonomia vehiculului au fost utile pentru evaluarea performanței vehiculului.

8. Anexă: Codul Sursă al Proiectului

```
package ace.ucv;

import java.util.HashMap;
import java.util.Map;

// Clasa de baza CadruVehicul pentru toate vehiculele
abstract class CadruVehicul {
    protected String nume;
    protected Map<String, Object> proprietati;

    public CadruVehicul(String nume, Map<String, Object> proprietatiInitiale) {
        this.nume = nume;
        this.proprietati = new HashMap<>(proprietatiInitiale);
    }

    public CadruVehicul(String nume) {
        this.nume = nume;
        this.proprietati = new HashMap<>();
    }

    public void adaugaProprietate(String numeProprietate, Object valoare) {
        proprietati.put(numeProprietate, valoare);
    }

    public Object obtineValoareProprietate(String numeProprietate) {
        Object valoare = proprietati.getOrDefault(numeProprietate, "undefined");
        if (valoare instanceof CalculatorProprietate) {
            return ((CalculatorProprietate) valoare).calcul();
        }
        return valoare;
    }
```

```
    public void afiseazaProprietati() {
        System.out.println("Proprietăți pentru " + nume + ":");
        for (String numeProprietate : proprietati.keySet()) {
            System.out.println(numeProprietate + ": " + obtineValoareProprietate(numeProprietate));
        }
    }
}

package ace.ucv;

// Interfata pentru calcularea atributelor dinamice
interface CalculatorProprietate {
    Object calcul();
}

// Nivelul 1: Vehicul
class Vehicul extends CadruVehicul {
    public Vehicul(Map<String, Object> proprietatiInitiale) {
        super("Vehicul", proprietatiInitiale);
    }

    public Vehicul() {
        super("Vehicul");
        adaugaProprietate("capacitate", "undefined");
        adaugaProprietate("viteza", "undefined");
    }
}

package ace.ucv;

import java.util.Map;

// Nivelul 2: VehiculMotorizat
class VehiculMotorizat extends Vehicul {
    public VehiculMotorizat(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
        adaugaProprietate("putereMotor", proprietatiInitiale.getOrDefault("putereMotor", 0));
        adaugaProprietate("calculVitezaMaxima", new CalculVitezaMaxima());
        adaugaProprietate("calculConsumCombustibil", new CalculConsumCombustibil());
        adaugaProprietate("calculAutonomie", new CalculAutonomie());
    }

    private class CalculVitezaMaxima implements CalculatorProprietate {
        public Object calcul() {
            int putereMotor = (Integer) obtineValoareProprietate("putereMotor");
            return putereMotor * 1.5;
        }
    }

    private class CalculConsumCombustibil implements CalculatorProprietate {
        public Object calcul() {
```

```
            int putereMotor = (Integer) obtineValoareProprietate("putereMotor");
            return putereMotor / 10.0;
        }
    }

    private class CalculAutonomie implements CalculatorProprietate {
        public Object calcul() {
            Object consumCombustibilObj = obtineValoareProprietate("calculConsumCombustibil");
            Object capacitateRezervorObj = obtineValoareProprietate("capacitateRezervor");

            // Verificam daca consumCombustibil si capacitateRezervor sunt de tipul corect
            if (consumCombustibilObj instanceof Double && capacitateRezervorObj instanceof Integer) {
                double consumCombustibil = (Double) consumCombustibilObj;
                int capacitateRezervor = (Integer) capacitateRezervorObj;
                return (capacitateRezervor / consumCombustibil) * 100;
            } else {
                return "undefined";
            }
        }
    }
}
```

```
package ace.ucv;

import java.util.Map;

//Nivelul 2
class VehiculNemotorizat extends Vehicul {
    public VehiculNemotorizat(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
    }
}
```

```
package ace.ucv;


import java.util.Map;

class VehiculDeTransport extends VehiculMotorizat {
    public VehiculDeTransport(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
        adaugaProprietate("utilizare", "transport");
    }
}
```

```
package ace.ucv;


import java.util.Map;

// Nivelul 3: VehiculPersonal
```

```
class VehiculPersonal extends VehiculMotorizat {
    public VehiculPersonal(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
        adaugaProprietate("utilizare", "personal");
    }
}

package ace.ucv;

import java.util.Map;

// Nivelul 4: Automobil
class Automobil extends VehiculPersonal {
    private Motor motor;
    private RezervorCombustibil rezervor;

    public Automobil(Map<String, Object> proprietatiMotor, Map<String, Object> proprietatiRezervor) {
        super(proprietatiMotor);
        this.motor = new Motor(proprietatiMotor);
        this.rezervor = new RezervorCombustibil(proprietatiRezervor);
        adaugaProprietate("tipCombustibil", motor.obtineCaracteristica("tipCombustibil"));
        adaugaProprietate("putereMotor", motor.obtineCaracteristica("putere"));
        adaugaProprietate("capacitateRezervor", rezervor.obtineCaracteristica("capacitate"));
    }
}


package ace.ucv;


import java.util.Map;

class Camion extends VehiculDeTransport {
    public Camion(Map<String, Object> proprietatiMotor, Map<String, Object> proprietatiRezervor) {
        super(proprietatiMotor);
        adaugaProprietate("tipCombustibil", proprietatiMotor.get("tipCombustibil"));
        adaugaProprietate("putereMotor", proprietatiMotor.get("putereMotor"));
        adaugaProprietate("capacitateRezervor", proprietatiRezervor.get("capacitateRezervor"));
        adaugaProprietate("capacitateIncarcare", 10000);
    }
}

package ace.ucv;

import java.util.Map;

// Nivelul 4: Bicicleta
class Bicicleta extends VehiculNemotorizat {
    public Bicicleta(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
        adaugaProprietate("tipCadru", proprietatiInitiale.getOrDefault("tipCadru", "Aluminiu"));
```

```
    }
}

package ace.ucv;

import java.util.Map;

// Nivelul 4: Clasa Motocicleta
class Motocicleta extends VehiculMotorizat {
    public Motocicleta(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
        adaugaProprietate("casca", proprietatiInitiale.getOrDefault("casca", true));
    }
}

package ace.ucv;

import java.util.HashMap;
import java.util.Map;

// Componentele de tip Agregare pentru Motor
class Motor {
    private Map<String, Object> caracteristici;

    public Motor(Map<String, Object> caracteristiciInitiale) {
        this.caracteristici = new HashMap<>(caracteristiciInitiale);
    }

    public Object obtineCaracteristica(String numeCaracteristica) {
        return caracteristici.getOrDefault(numeCaracteristica, "undefined");
    }
}

package ace.ucv;

import java.util.HashMap;
import java.util.Map;

// Componentele de tip Agregare pentru  RezervorCombustibil
class RezervorCombustibil {
    private Map<String, Object> caracteristici;

    public RezervorCombustibil(Map<String, Object> caracteristiciInitiale) {
        this.caracteristici = new HashMap<>(caracteristiciInitiale);
    }

    public Object obtineCaracteristica(String numeCaracteristica) {
        return caracteristici.getOrDefault(numeCaracteristica, "undefined");
    }
}
```

```
class Trotineta extends VehiculNemotorizat {
    public Trotineta(Map<String, Object> proprietatiInitiale) {
        super(proprietatiInitiale);
        adaugaProprietate("esteElectrica", proprietatiInitiale.getOrDefault("esteElectrica", true));
        adaugaProprietate("calculVitezaMaxima", new CalculVitezaMaxima());
    }

    private static class CalculVitezaMaxima implements CalculatorProprietate {
        public Object calcul() {
            boolean esteElectrica = true; // valoare exemplu
            return esteElectrica ? 25 : 15;
        }
    }
}
package ace.ucv;

import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        // Definirea caracteristicilor pentru Motor si Rezervor
        Map<String, Object> proprietatiMotor = new HashMap<>();
        proprietatiMotor.put("putere", 250);
        proprietatiMotor.put("tipCombustibil", "Diesel");

        Map<String, Object> proprietatiRezervor = new HashMap<>();
        proprietatiRezervor.put("capacitate", 60);

        // Crearea unei instante de Automobil care este un VehiculPersonal
        Automobil masinaFamilie = new Automobil(proprietatiMotor, proprietatiRezervor);

        // Caracteristicile pentru Camion
        Map<String, Object> proprietatiMotorCamion = new HashMap<>();
        proprietatiMotorCamion.put("putereMotor", 400);
        proprietatiMotorCamion.put("tipCombustibil", "Motorina");

        Map<String, Object> proprietatiRezervorCamion = new HashMap<>();
        proprietatiRezervorCamion.put("capacitateRezervor", 120);

        // Instanta de Camion
        Camion camionMarfa = new Camion(proprietatiMotorCamion, proprietatiRezervorCamion);

        // Instanta de Bicicleta
        Map<String, Object> proprietatiBicicleta = new HashMap<>();
        proprietatiBicicleta.put("tipCadru", "Carbon");
        Bicicleta bicicletaMountain = new Bicicleta(proprietatiBicicleta);

        // Instanta de Trotineta
        Map<String, Object> proprietatiTrotineta = new HashMap<>();
        proprietatiTrotineta.put("esteElectrica", true);
```

```
        Trotineta trotinetaElectrica = new Trotineta(proprietatiTrotineta);

        // Instanta de Motocicleta
        Map<String, Object> proprietatiMotocicleta = new HashMap<>();
        proprietatiMotocicleta.put("putereMotor", 10);
        proprietatiMotocicleta.put("tipCombustibil", "Benzina");
        proprietatiMotocicleta.put("casca", true);

        // Instanta de Motocicleta
        Motocicleta motocicletaSport = new Motocicleta(proprietatiMotocicleta);

        System.out.println("Detalii Automobil - Masina de Familie:");
        masinaFamilie.afiseazaProprietati();

        System.out.println("\nDetalii Bicicleta - Bicicleta Mountain:");
        bicicletaMountain.afiseazaProprietati();

        System.out.println("\nDetalii Trotineta - Trotineta Electrica:");
        trotinetaElectrica.afiseazaProprietati();


        System.out.println("\nDetalii Camion - Camion de Marfa:");
        camionMarfa.afiseazaProprietati();

        System.out.println("\nDetalii Motocicleta - Motocicleta Sport:");
        motocicletaSport.afiseazaProprietati();

    }
}
```