# CREAREA DINAMICĂ A CLASELOR ÎN CONTEXTUL AUTOMATIZĂRII PROCESULUI DE DEZVOLTARE

**Vitalii ARMAȘ[1*], Arina CARAUȘ[2], Cristian PÎRLEA[2], Sergiu BALTEAN[2], Marian TABUREANU[2]**

*[1]Matematică şi Informatică, Grupa 331, Universitatea Transilvania din Braşov, Braşov, România*
*[2]Informatică şi Ingineria Sistemelor, CR-212, Calculatoare, Informatică si Microelectronică, Universitatea Tehnică din Moldova, Chişinău, Republica Moldova*

*Autorul corespondent: Vitalii ARMAȘ, vitalii.armas@student.unitbv.ro

Coordonator științific: **Lilia ROTARU**, lector universitar, DIIS

***Rezumat.*** *Acest articol este o investigare teoretică a procesului de creare dinamică a claselor și a importanței inițierii acestora în momentul de run-time pentru optimizarea dezvoltării aplicațiilor. Subiectul este abordat din perspectiva paradigmei „Programarea Orientată pe Obiecte". În continuare, sunt indicate metodele folosirii acestui concept, cu scopul de a crea o imagine clară asupra utilizării claselor dinamice în diferite limbaje şi pentru a înainta raționamentele care vor indica un randament maxim al programului implementat. Menționarea combinării algoritmilor genetici cu acest concept, permite înțelegerea abordării promițătoare pentru crearea dinamică a claselor pe baza principiilor de optimizare. De asemenea, sunt prezentate domeniile de utilizare, modul de implementare, și forma de abstractizare a soluțiilor dinamice oferite societății. Prin intermediul unui use case este indicată aplicabilitatea și relevanța implementării acestui concept pentru ajustarea rezultatului conform contextului și datelor obținute în timp real. Totodată, prezentăm etapele ce reprezintă intercalarea creării dinamice a claselor cu algoritmii genetici pentru a ușura înțelegerea acestora.*

***Cuvinte cheie:*** *gene, execuție, adaptare, POO.*

## Introducere

Metodologiile de programare sunt cadre cruciale utilizate de dezvoltatori pentru a aborda provocări complexe din lumea reală și pentru a organiza în mod eficient dezvoltarea software-ului. Aceste metodologii variază în mod semnificativ în ceea ce privește strategiile lor de descompunere a problemelor și de organizare structurală [1].

Programarea orientată pe obiecte (POO) adoptă o abordare diferită, concentrându-se în jurul unor entități sau obiecte inerente domeniului problemei. Soluția este structurată în jurul acestor entități, punând accentul pe datele, comportamentele și interacțiunile acestora pentru a crea o soluție unificată și coerentă. Această metodologie îmbunătățește capacitatea de reutilizare, scalabilitate și mentenanță a codului prin încapsularea datelor și metodelor în obiecte [2].

Generarea dinamică a claselor se referă la capacitatea de a crea clase chiar în timpul execuției, fără a fi nevoie să le definiți în mod explicit în cod. Acest lucru poate fi util într-o varietate de situații, cum ar fi atunci când trebuie să creați un număr mare de clase similare sau când doriți să personalizați comportamentul unei clase pe baza informațiilor din timpul execuției.

În contextul programării orientate pe obiecte, în special în ceea ce privește structurarea meticuloasă a soluțiilor în jurul entităților sau obiectelor și accentul pus pe încapsulare, se poate aplica direct la generarea dinamică de clase [2]. POO oferă mecanisme precum moștenirea, polimorfismul și reflecția, care permit dezvoltatorilor să creeze și să manipuleze în mod dinamic clasele în timpul execuției, pe baza unor cerințe specifice [2]. Această flexibilitate permite crearea unor soluții modulare, dinamice și adaptabile, în conformitate cu principiile POO.

Actualitatea automatizării codării folosind crearea dinamică a claselor constă în potențialul său de a simplifica și îmbunătăți procesul de dezvoltare, în special în scenariile în

care predomină sarcinile repetitive sau comportamentele dinamice. Iată câteva puncte cheie care evidențiază importanța automatizării codării prin intermediul creării dinamice a claselor:

- Eficiență: Crearea dinamică a claselor automatizează procesul de generare a claselor din mers, eliminând nevoia de codificare manuală a definițiilor de clasă. Acest lucru economisește timp și efort de dezvoltare semnificativ, în special în situațiile în care trebuie creat un număr mare de clase similare sau când sunt necesare modificări frecvente.
- Flexibilitate: Automatizarea prin crearea dinamică a claselor oferă flexibilitate în adaptarea la cerințele în schimbare și la condițiile de execuție [2]. Dezvoltatorii pot crea clase în mod dinamic pe baza unor informații specifice din timpul execuției, permițând soluții software mai adaptative și mai receptive.
- Scalabilitate: Crearea dinamică a claselor facilitează scalabilitatea, permițând crearea de clase în timpul execuției în funcție de cerere. Această scalabilitate este benefică în cazul aplicațiilor cu cerințe în evoluție sau în cazul sistemelor care gestionează volume mari de date.
- Personalizare: Automatizarea prin crearea dinamică a claselor permite dezvoltatorilor să personalizeze comportamentul claselor în baza parametrilor de execuție sau a intrărilor utilizatorului. Acest nivel de personalizare sporește adaptabilitatea și versatilitatea software-ului [2].
- Întreținere: Generarea automatizată de clase poate îmbunătăți mentenabilitatea codului prin centralizarea logicii de creare a claselor și reducerea duplicării. Modificările sau actualizările comportamentului clasei pot fi aplicate uniform în clasele generate dinamic, simplificând eforturile de întreținere.

**Descrierea creării dinamice**

Crearea dinamică a claselor este caracteristica limbajelor de programare ce suportă programarea orientată pe obiecte. Astfel, ne-am pus ca scop să cercetam metodele ei de implementare în limbajele Python și JavaScript.

**JavaScript**

JavaScript (JS) este un limbaj interpretabil orientat pe obiecte. Cu toate că sintaxa acestui limbaj seamănă cu C++ sau Java, JS nu este un limbaj puternic tipizat, ceea ce însemna că variabilele nu au neapărat un tip specificat. În acest caz, JS seamănă mai mult cu Perl, iar mecanismul său de moștenire este bazat pe prototipe, ca în limbajul Self [3]. Crearea dinamică a claselor în acest limbaj poate fi realizată prin intermediul obiectelor de tip JSON [4] și a metodei Object.assign [5].

Pentru început, se creează un obiect JSON care va conține informațiile necesare pentru a crea dinamic o clasă, după cum urmează în figura 1:

```
let classInfo = {
    className: "Persoana",
properties: {
nume: "",
varsta: 0,
    },
methods: {
salut: function() {
return "Salut, eu sunt " + this.nume + " si am " + this.varsta + " ani.";
}
}
    };
```

**Figura 1. Crearea obiectului JSON**

În acest caz, obiectul JSON pe care îl vom crea se numește "classInfo", acesta conține numele viitoarei clase "Persoana", un obiect "properties" cu viitoarele câmpuri ale clasei: "nume" și "varsta" și un obiect "methods" ce conține metoda viitoarei clase, "greet".

După crearea obiectului JSON, definim o funcţie “creazaDinamicClasa” ca şi în figura 2:

```
function creazaDinamicClasa(classInfo) {

    let clasaDinamica = function(data) {

        Object.assign(this, data);
    };

    Object.assign(clasaDinamica.prototype, classInfo.properties, classInfo.methods);

    return clasaDinamica;
}
```

**Figura 2. Funcţia creazaDinamicClasa**

Aceasta funcţie primeşte obiectul JSON “classInfo” ca parametru și definește un constructor “clasaDinamica” care va primi ca parametru un obiect “data”. Prin intermediul metodei Object.assign() constructorul va asigna proprietățile din obiectul “data” la instanța curentă (“this”) [3,5]. În final, cu ajutorul aceleiași metode, se adaugă proprietățile și metodele din “classInfo” la prototipul clasei “dynamicClass”.

Următorul pas este crearea dinamică şi utilizarea clasei (figura 3):

```
let Persoana = creazaDinamicClasa(classInfo);

let persoana1 = new Persoana({ nume: "Ion", varsta: 30 });

console.log(person1.salut());
```

**Figura 3. Crearea dinamică și utilizarea clasei**

Aici se creează, în timpul execuției, clasa nouă „Persoana” prin intermediul funcției definite anterior și a obiectului JSON, după care, cu ajutorul constructorului se inițializează un obiect „persoana1” de tipul clasei noi cu parametrii „nume” și „varsta”, iar în final se apelează metoda „salut()” a obiectului. În rezultatul compilării în consolă se afișează mesajul „Salut, eu sunt Ion si am 30 ani.”.

**Python**

Python este unul dintre cele mai populare limbaje de programare din lume. Unul dintre aspectele care îl distinge este sistemul său de tipizare puternică [6], ce permite programatorilor să creeze și să modifice clase într-un mod dinamic și eficient. Crearea dinamică în Python se poate face în mai multe moduri.

Prima metoda este utilizarea funcției type ce poate primi 3 argumente: denumirea clasei, clasele pe care le va moșteni noua clasă și dicționarul ce va conține atributele și metodele (sub formă de obiecte) ale viitoarei clase [7], după cum e prezentat și în figura 4.

```
ClasaNoua = type('ClasaNoua', (object, ), dict())
```

**Figura 4. Crearea dinamică a clasei vide ClasaNoua**

În aceasta figură se ilustrează crearea unei clase vide cu denumirea “ClasaNoua”, ce moşteneşte clasa object, dar care nu are atribute sau metode, deoarece dicționarul este vid. Se pot adaugă în dicționar obiecte ce vor reprezenta metode sau denumiri ale unor variabile care vor reprezenta câmpurile noii clase.

O altă metodă de creare dinamică este utilizarea metaclaselor, și anume, prin moștenirea metaclasei "type". Se definește o clasă nouă care moștenește metaclasa "type". În cadrul acestei clase, se suprascrie metoda "__new__()" ce are ca parametri de intrare metaclasa, numele clasei ce urmează să fie creată, clasele pe care aceasta le moștenește și dicționarul de atribute ce va conține atributele și metodele clasei noi [7]. Un exemplu de cod este ilustrat în figura 5.

```
class myMeta(type):
    def __new__(cls, name, bases, dct):

        dct['metodaDinamica'] = lambda self: print("Aceasta este o metoda dinamica.")

        return super().__new__(cls, name, bases, dct)

clasaDinamica = myMeta('clasaDinamica', (), {})

ob1 = clasaDinamica()
ob1.metodaDinamica()
```

**Figura 5. Crearea dinamică a claselor folosind metaclasa "type"**

Se definește clasa "myMeta" care va moșteni clasa "type", devenind astfel și ea o metaclasă. În interiorul acestei clase se suprascrie metoda "__new__()" care va adăuga în viitoarea clasă o funcție lambda ce va afișa în consolă un mesaj. Prin intermediul clasei "myMeta" și a parametrilor de intrare: cls se transmite implicit, 'clasaDinamica' reprezintă numele clasei, () reprezintă clasele pe care le va moșteni vitoarea clasă (în cazul nostru, nici una) și {} reprezintă atributele și metodele clasei noi (în exemplul dat, clasa nouă nu va avea atribute sau metode).

**Studiul de caz**

În următorul use-case este utilizată crearea dinamică a claselor pentru automatizarea procesului și oferirea posibilității utilizatorului de a influența codul sursă, în timpul execuției.

Inițial, în program se definește clasa de bază 'Părinte', care deține o metodă pentru transmiterea trăsăturilor genetice.

```
class Parinte:
    def __init__(self, denumire, trasaturi):
        self.denumire = denumire
        self.trasaturi = trasaturi

    def transmite_genetic(self):
        return self.trasaturi
```

**Figura 6. Crearea clasei Parinte**

După care se inițiază o funcție pentru contopirea genelor, care primește trăsăturile părinților de la tastatură. Conținutul introdus de utilizator în campurile entry_parinte1_trasaturi și entry_parinte2_trasaturi, este interpretat și convertit în obiecte dicționar, utilizând eval(). Această operație este necesară pentru a putea fi prelucrate în continuare în funcția dată.

```
def contopire_gene():
    parinte1_denumire = entry_parinte1_denumire.get()
    parinte2_denumire = entry_parinte2_denumire.get()
    parinte1_trasaturi = eval(entry_parinte1_trasaturi.get())
    parinte2_trasaturi = eval(entry_parinte2_trasaturi.get())
```

**Figura 7. Metoda de contopire a genelor**

Sunt create 2 obiecte de tipul clasei ‘Părinte’, utilizând datele recepţionate.

```
parinte1 = Parinte(parinte1_denumire, parinte1_trasaturi)
parinte2 = Parinte(parinte2_denumire, parinte2_trasaturi)
```

**Figura 8. Crearea obiectelor clasei ‘Părinte’**

Următorul pas este creare dinamică a clasei ‘Copil’, folosind type(), care primește denumirea clasei (‘Copil’), clasele părinţi (Parinte), și un dicționar cu metodele și atributele clasei. Metoda ‘__init__’ este suprascrisă pentru a inițializa trăsăturile copilului cu valorile calculate anterior.

```
try:
    Copil = type('Copil', (Parinte,), {'__init__': lambda self:
    Parinte.__init__(self, f"{parinte1_denumire}{parinte2_denumire}",
    trasaturi_copil)})
except Exception as e:
    messagebox.showerror("Eroare", f"Eroare la crearea clasei Copil: {e}")
    return
```

**Figura 9. Crearea clasei copil folosind type()**

Ulterior utilizatorul poate adăuga funcții noi, sau modifica funcțiile deja existente, la moștenirea dinamică a clasei copil, utilizând funcţia exec(), care este folosită pentru a executa codul introdus de la tastatură. Acest lucru permite utilizatorului să adauge funcționalități personalizate la clasa copil, fără a fi nevoie să modifice direct codul sursă al aplicației.

```
try:
    exec(entry_parinte_custom.get())
except Exception as e:
    messagebox.showerror("Eroare", f"Eroare la execuția codului custom:
    {e}")
    return
```

**Figura 10. Adăugarea altor funcții în moştenirea dinamică**

Prin automatizarea procesului de creare a claselor copil în funcție de trăsăturile părinților, acest use case contribuie la eficientizarea procesului de dezvoltare și la flexibilitatea în gestionarea diversității trăsăturilor genetice.

**Concluzii**

Crearea dinamică a claselor este o unealtă promițătoare în condițiile în care scrierea unui cod devine un proces tot mai complex și din ce în ce mai monoton. Automatizarea etapei de dezvoltare în contextul programării dinamice reprezintă o direcție importantă pentru evoluția

procesului de creare a aplicațiilor, care poate fi aplicată cu success în diferite limbaje (precum Python sau Java Script) în tandem cu inteligența artificială. Aceasta poate sta la baza dezvoltării aplicațiilor dinamice utilizând metodologia funcțional-Incrementară, permițând ajustarea sau introducerea cerințelor noi pe parcursul procesului de dezvoltare și oferă posibilitatea de a elimina erorile în cazul în care acestea pot apărea la etapele de analiză sau proiectare. Astfel metodologia dată de dezvoltare oferă oportunitatea de a ajusta funcționalitățile existente și adăuga unele noi. Un dezavantaj ar putea fi impredictibilitatea direcției de evoluție a aplicației, risc ce ar putea fi redus prin introducerea unor restricții bine definite. Rezultatele obtinute în urma execuției programului din studiul de caz confirmă aplicabilitatea metodei descrise si demonstrează potențialul oferit de combinarea acesteia cu inteligența artificială în scrierea de cod și procesul de dezvoltare a aplicațiilor software. Programarea dinamică și în special definirea claselor și moștenirea dinamică este un instrument puternic ce poate fi folosit în diferite domenii, precum: Framework-uri (unele framework-uri pentru web, cum ar fi Django sau Flask în Python, pot beneficia de crearea dinamică a claselor pentru a gestiona rutele și logica aplicației într-un mod flexibil și modular), sau chiar în dezvoltarea aplicațiilor inspirate din natură (inteligența artificială, algoritmii genetici, fuzzi/neuro fuzzy). Îmbinându-se cu sistemele de procesare a limbajului natural, această tehnologie poate accelera procesul de dezvoltare a software-ului pentru ingineri și programatori, iar perspectivele utilizării lor pot fi evaluate ca fiind promițătoare.

**Referințe**

[1] Annabelle MckIver, Carroll Morgan, Programming Methodology, Springer-Verlag New York, DOI: 10.1007/978-0-387-21798-7, 2003.

[2] Mark Lutz, *Learning Python: Powerful Object-Oriented Programming.* O'Reilly Media, ISBN: 978-1-449-35573-9, 2013.

[3] David Flanagan, *JavaScript The Definitive Guide.* O'Reilly Media, ISBN: 9781491952023, 2020.

[4] Mateusz Piech, Robert Marcjan *A new approach to storing dynamic data in relational databases using json.* DOI: https://doi.org/10.7494/csci.2018.19.1.2505, 2018.

[5] Kyle Simpson, *You Don't Know JS: this & Object Prototypes.* O'Reilly Media, ISBN: 978-1-491-90415-2, 2014.

[6] John Hunt, *A Beginners Guide to Python 3 Programming.* Springer Nature Switzerland AG, ISBN 978-3-030-20289-7, 2020.

[7] John Hunt, *Advanced Guide to Python 3 Programming.* Springer Nature Switzerland AG, ISBN 978-3-030-25942-6, 2023.