



Curs java pentru testare automata

JAVA

Cuprins curs:

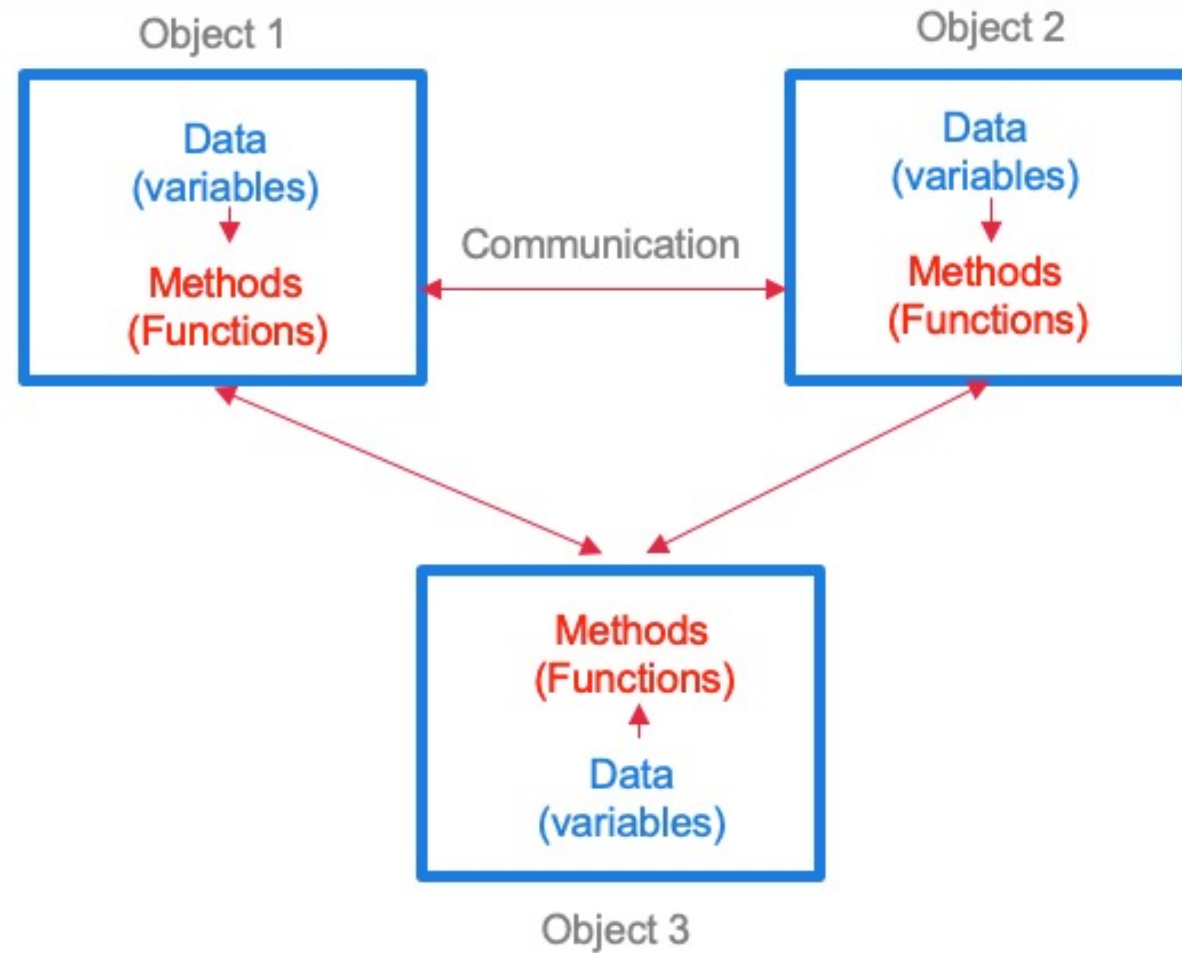
- ▶ Declarare si initializare obiecte
- ▶ Life cycle al unui obiect
- ▶ Anonymus object
- ▶ Constructor

OOP (Object Oriented Programming)

- ▶ Programarea orientată pe obiecte (OOP) în Java este o metodologie de programare sau paradigmă (model) pentru a proiecta un program de calculator folosind clase și obiecte.
- ▶ Este cea mai populară paradigmă de programare și utilizată pe scară largă în industria software-ului de astăzi. Este o extensie a programării procedurale.
- ▶ Programarea procedurală înseamnă scrierea codului fără obiecte, în timp ce programarea orientată pe obiecte înseamnă scrierea codului cu obiecte care conțin date sub formă de câmpuri sau variabile (adesea cunoscute ca attribute sau proprietăți în java) și cod de funcționalitate sub formă de metode (adesea cunoscute sub numele de comportament în java).

OOP (Object Oriented Programming)

- ▶ Conceptul Java OOP folosește variabile și metode așa cum fac programele procedurale, dar se concentrează pe obiectele care conțin variabile și metode.
- ▶ De aceea această nouă abordare se numește abordare orientată pe obiecte, în care scriem programe folosind clase și obiecte.
- ▶ Acesta abordare vine cu patru caracteristici principale:
 - ▶ Encapsulation (încapsularea)
 - ▶ Abstraction (abstractizarea)
 - ▶ Inheritance (moștenirea)
 - ▶ Polymorphism (polimorfismul)



OOP (Object Oriented Programming)

- ▶ Datele unui obiect pot fi accesate și modificate numai prin funcția acelui obiect.
- ▶ O funcție externă nu poate accesa datele unui obiect, dar funcțiile unui obiect pot accesa funcția altor obiecte.
- ▶ Astfel, datele sunt protejate de modificări prin funcții externe. În acest fel, datele sunt ascunse în limbajul de programare orientat pe obiecte.

OOP : Object

- ▶ O entitate care are stare și comportament se numește obiect în java.
- ▶ Aici, starea reprezintă proprietăți, iar comportamentul reprezintă acțiuni și funcționalitate.
- ▶ De exemplu, o persoană, scaun, stilou, masă, tastatură, bicicletă etc.
- ▶ Să luăm un exemplu real pentru a înțelege caracteristicile unui obiect. O persoană are trei caracteristici:
 - ▶ Identitate (nume)
 - ▶ Stare (proprietăți)
 - ▶ Comportament (acțiuni sau funcționalitate).

OOP : Object

- ▶ Starea / proprietățile unei persoane sunt:
 - ▶ părul negru, ochii negri, pielea albă, înălțimea de 180 cm.
- ▶ Acțiunile sau comportamentul persoanei poate fi :
 - ▶ mânâncă, doarme, merge, învață

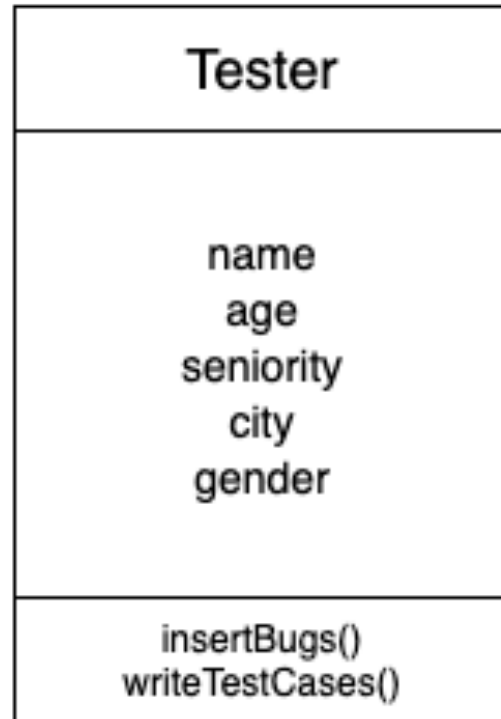
OOP : Class

- ▶ O clasă este practic un tip de date definit de utilizator care acționează ca un șablon pentru crearea obiectelor de tip identic.
- ▶ Reprezintă proprietățile și acțiunile (funcțiile) comune ale unui obiect.
- ▶ De exemplu, autobuzul și mașina sunt obiecte din clasa vehiculelor. Vrabia și papagalul sunt obiecte din clasa păsărilor.

Functionality

Attributes

Class



name = Matei
age = 24
seniority = Junior
city = Iasi
gender = M

name = Oana
age = 30
seniority = Mid
city = Cluj
gender = F


name = Bobo
age = 35
seniority = Senior
city = Ploiesti
gender = M

Sign up

* Name:

* Last Name:

* Email: 

* Password: 

* Confirm:

```
public class UserRegistration {  
  
    private String Name;  
    private String LastName;  
    private String Email;  
    private String Password;  
  
    public void saveUserToDatabase() {  
    }  
  
    public void checkEmailUniqness() {  
  
    }  
  
}
```


Square Footage Calculator

Area Shape: Rectangle

[Rectangle Image](#)

Length = ft

Width = ft

Quantity =

Optional Cost Calculation

\$ price per square foot (ft²)

```
public class Rectangle {  
  
    double lenght;  
    double width;  
  
    double calculeazaPerimetru() {  
        return (2 * lenght) + (2 * width);  
    }  
  
    double calculeazaAria() {  
        return lenght * width;  
    }  
  
}
```


Object

- ▶ Crearea unui obiect se mai numește instanțierea unui obiect. Prin crearea unui obiect, putem accesa membrii clasei din care a fost creat obiectul.

```
public class Rectangle {  
    double lenght;  
    double width;  
  
    double calculeazaPerimetru() {  
        return (2 * lenght) + (2 * width);  
    }  
  
    double calculeazaAria() {  
        return lenght * width;  
    }  
}
```

Object Rectangle 1

lenght = 4
width = 2

Object Rectangle 2

lenght = 5
width = 3

Object Rectangle 3

lenght = 3
width = 2


```
public class TestRectangle {  
  
    public static void main(String[] args) {  
  
        //Rectangle 1  
        Rectangle rectangle1 = new Rectangle();  
        rectangle1.setLenght(4.0);  
        rectangle1.setWidth(2.0);  
  
        //Rectangle 2  
        Rectangle rectangle2 = new Rectangle();  
        rectangle2.setLenght(5.0);  
        rectangle2.setWidth(3.0);  
  
        //Rectangle 3  
        Rectangle rectangle3 = new Rectangle();  
        rectangle3.setLenght(3.0);  
        rectangle3.setWidth(2.0);  
    }  
}
```

```
public class Rectangle {  
  
    double lenght;  
    double width;  
  
    public double getLenght() {  
        return lenght;  
    }  
  
    public void setLenght(double lenght) {  
        this.lenght = lenght;  
    }  
  
    public double getWidth() {  
        return width;  
    }  
  
    public void setWidth(double width) {  
        this.width = width;  
    }  
}
```

```
public class TestRectangle {  
  
    public static void main(String[] args) {
```

```
        //Rectangle 1
```

```
        Rectangle rectangle1 = new Rectangle();  
        rectangle1.setLenght(4.0);  
        rectangle1.setWidth(2.0);
```

```
        //Rectangle 2
```

```
        Rectangle rectangle2 = new Rectangle();  
        rectangle2.setLenght(5.0);  
        rectangle2.setWidth(3.0);
```

```
        //Rectangle 3
```

```
        Rectangle rectangle3 = new Rectangle();  
        rectangle3.setLenght(3.0);  
        rectangle3.setWidth(2.0);
```

```
public class Rectangle {
```

```
    double lenght;  
    double width;
```

```
    public double getLenght() {  
        return lenght;  
    }
```

```
    public void setLenght(double lenght) {  
        this.lenght = lenght;  
    }
```

```
    public double getWidth() {  
        return width;  
    }
```

```
    public void setWidth(double width) {  
        this.width = width;  
    }
```

```
}
```



```
public class TestRectangle {  
  
    public static void main(String[] args) {  
  
        //Rectangle 1  
        Rectangle rectangle1 = new Rectangle();  
        rectangle1.setLenght(4.0);  
        rectangle1.setWidth(2.0);  
  
        //Rectangle 2  
        Rectangle rectangle2 = new Rectangle();  
        rectangle2.setLenght(5.0);  
        rectangle2.setWidth(3.0);  
  
        //Rectangle 3  
        Rectangle rectangle3 = new Rectangle();  
        rectangle3.setLenght(3.0);  
        rectangle3.setWidth(2.0);  
    }  
}
```

```
public class Rectangle {  
  
    double lenght;  
    double width;  
  
    public double getLenght() {  
        return lenght;  
    }  
  
    public void setLenght(double lenght) {  
        this.lenght = lenght;  
    }  
  
    public double getWidth() {  
        return width;  
    }  
  
    public void setWidth(double width) {  
        this.width = width;  
    }  
}
```

```
public class Rectangle {
```

```
    double lenght;  
    double width;
```

```
    public double getLenght() {  
        return lenght;  
    }
```

```
    public void setLenght(double lenght) {  
        this.lenght = lenght;  
    }
```

```
    public double getWidth() {  
        return width;  
    }
```

```
    public void setWidth(double width) {  
        this.width = width;  
    }
```

```
}
```

```
public class TestRectangle {
```

```
    public static void main(String[] args) {
```

```
        //Rectangle 1
```

```
        Rectangle rectangle1 = new Rectangle();  
        rectangle1.setLenght(4.0);  
        rectangle1.setWidth(2.0);
```

```
        //Rectangle 2
```

```
        Rectangle rectangle2 = new Rectangle();  
        rectangle2.setLenght(5.0);  
        rectangle2.setWidth(3.0);
```

```
        //Rectangle 3
```

```
        Rectangle rectangle3 = new Rectangle();  
        rectangle3.setLenght(3.0);  
        rectangle3.setWidth(2.0);
```

```
    }
```

lenght = 4
width = 2

lenght = 5
width = 3

lenght = 3
width = 2

Object

- ▶ In Java, un obiect al unei clase este creat folosind cuvântul cheie **'new'**.
- ▶ Sunt trei pași care se întâmplă după cum urmează:
- ▶ 1. Declarația unei variabile de referință.
- ▶ 2. Crearea unui obiect.
- ▶ 3. Legarea obiectului și a variabilei de referință.

```
public class Circle {  
  
}
```


Object

- ▶ Sunt trei pași care se întâmplă după cum urmează:
- ▶ 1. Declarația unei variabile de referință.
- ▶ **Rectangle dreptunghi;**
- ▶ 2. Crearea unui obiect.
- ▶ **new Rectangle();**
- ▶ 3. Legarea obiectului și a variabilei de referință.
- ▶ **=**
- ▶ **Rectangle dreptunghi = new Rectangle();**

Object

- ▶ Sunt trei pași care se întâmplă după cum urmează:
- ▶ 1. Declarația unei variabile de referință.
- ▶ *Circle cerc;*
- ▶ 2. Crearea unui obiect.
- ▶ *new Circle();*
- ▶ 3. Legarea obiectului și a variabilei de referință.
- ▶ *=*
- ▶ *Circle cerc = new Circle();*

Life cycle of an Object

- ▶ Ar trebui să știm cum se nasc obiecte, își trăiesc viața și apoi mor la sfârșit.
- ▶ Există următorii pași care apar în ciclul de viață al obiectului în java:
- ▶ Pasul 1: Crearea fișierului .class pe disc După cum știți, programele Java rulează pe mașina virtuală Java (JVM). Când compilăm clasa Java, aceasta este transformată în bytecode, care este independent de platformă și de mașină. Clasele compilate sunt stocate ca fișier .class pe disc.
- ▶ Pasul 2: Încărcarea fișierului .class în memorie După aceea, runtime-ul Java află de acea clasă de pe disc care se află sub forma unui fișier .class. Java Classloader încarcă această clasă în memorie și apoi Java runtime o citește în memorie.

Life cycle of an Object

- ▶ Pasul 3: Căutarea membrilor statici inițiali ai clasei Acum, Java caută toți membrii statici inițializați ai clasei, cum ar fi metoda statică, câmpul static și blocul static. Vă amintiți întotdeauna că toți membrii statici ai clasei nu aparțin niciunei instanțe particulare a clasei. Acesta aparține clasei în sine și este împărtășit de toate obiectele create din clasă.
- ▶ Pasul 4: Initializarea. În primul rând, când accesați un câmp static sau o metodă statică a clasei. De exemplu, atunci când executați metoda principală într-o clasă, clasa este inițializată deoarece metoda principală este statică, iar a doua modalitate este atunci când obiectul sau instanța clasei este creat folosind cuvântul cheie NEW, clasa este inițializată.

Life cycle of an Object

- ▶ Pasul 5: Alocarea memoriei pentru obiect și variabilă de referință .Java alocă memoria de pe heap pentru obiect și memoria stack pentru variabila de referință a obiectului.
- ▶ Pasul 6: Apelarea constructorului clasei După alocarea memoriei, JVM apelează constructorul clasei care este ca o metodă, dar este apelat o singură dată când obiectul este creat.
- ▶ Astfel, obiectul își trăiește viața și oferă acces la câmpurile și metodele sale orice ne dorim și avem nevoie pentru a le accesa.
- ▶ Pasul 7: Eliminarea obiectului și variabilei de referință din memorie Când accesarea câmpului și a metodelor sunt finalizate, obiectul și referința acestuia sunt eliminate din memorie de către JVM. În acest moment obiectul a murit. Nu trebuie să distrugeți singur obiecte. De fapt, atunci când obiectul nu mai este utilizat, runtime-ul Java apelează colectorul de gunoi (garbage collector) pentru a distruge toate obiectele.
- ▶ Astfel, obiectele se nasc, trăiesc și mor.

Anonymous object

- ▶ Un obiect care nu are nicio variabilă de referință se numește obiect anonim în Java.
- ▶ Anonim înseamnă fără nume. Dacă doriți să creați un singur obiect într-o clasă, atunci obiectul anonim este o abordare bună.
- ▶ Syntax:
`new class_Name();`

Java Constructor

- ▶ Un constructor în java este un bloc de cod, similar cu o metodă care este utilizată pentru a inițializa starea unui obiect dintr-o clasă prin operatorul NEW.
- ▶ Constructorul Java poate efectua orice acțiune, dar special conceput pentru a efectua acțiuni de inițializare, cum ar fi inițializarea variabilelor de instanță.
- ▶ Un constructor dintr-o clasă permite construirea obiectului clasei în timpul rulării. Este invocat când se creează o instanță a unei clase folosind NEW operator.
- ▶ Constructorii pot accepta, de asemenea, argumente precum metode și pot fi supraîncărcați. Dacă încercăm să creăm un obiect al clasei fără a specifica vreun constructor, JVM va crea un constructor pentru noi.

Java Constructor

- ▶ Există următoarele caracteristici ale constructorului în java:
- ▶ 1. Numele constructorului trebuie să fie același cu numele clasei în care este declarat și definit.
- ▶ 2. Constructorul nu ar trebui să aibă niciun tip de returnare chiar nul, deoarece, dacă există un tip de returnare, atunci JVM ar considera o metodă, nu un constructor. Compilatorul și JVM diferențiază definițiile constructorului și ale metodelor pe baza tipului de returnare. Să presupunem că definiți metoda și constructorul cu același nume ca cel al numelui clasei, atunci JVM ar face diferența între ele utilizând tipul return.

Java Constructor

- ▶ 3. Ori de câte ori creăm un obiect / instanță al unei clase, constructorul va fi apelat automat de către JVM (Java Virtual Machine).
- ▶ Dacă nu definim niciun constructor în interiorul clasei, **compilatorul Java creează automat un constructor implicit în timpul compilării și atribuie valori implicite pentru toate variabilele declarate în clasă.**
- ▶ Valorile implicite pentru variabile sunt după cum urmează:
 - ▶ A. Variabilele numerice sunt setate la 0.
 - ▶ B. Șirurile sunt setate la nul.
 - ▶ C. Variabilele booleene sunt setate la false.

Java Constructor

- ▶ 4. Constructorul Java poate conține sau nu parametri. Parametrii sunt variabile locale pentru a primi valoare (date) din exterior într-un constructor.
- ▶ 5. Un constructor este apelat și executat automat de JVM în momentul creării obiectului. JVM alocă mai întâi memoria pentru variabile (obiecte) și apoi execută constructorul pentru a inițializa variabilele de instanță.
- ▶ 6. Este apelat și executat o singură dată pe obiect. Când creăm al doilea obiect, atunci constructorul este apelat din nou în timpul celui de-al doilea timp.

Java Constructor

- ▶ Practic, există două tipuri de constructori în java. :
- ▶ Constructor implicit (constructor fără argumente)
- ▶ Constructor parametrizat (constructor de argumente)

Structura unei clase

- ▶ **Body** : Continutul unei clase sta intre o pereche de {}
- ▶ **O clasa poate contine urmatoarele elemente :**
- ▶ **Fields**: Câmpurile sunt variabile membre ale unei clase care stochează date sau valori în ea. Specifică starea sau proprietățile clasei și obiectul acesteia. O variabilă poate fi o variabilă locală, variabilă de instanță sau variabilă statică.
- ▶ **Constructor**: Un constructor este utilizat pentru a crea un obiect. Fiecare clasă trebuie să conțină cel puțin un constructor în caz contrar, niciun obiect nu poate fi creat din clasă. Dacă nu specificăm un constructor, se alocă unul default.

Structura unei clase

- ▶ **Body** : Continutul unei clase sta intre o pereche de {}
- ▶ **O clasa poate contine urmatoarele elemente :**
- ▶ **Methods:** o metodă definește acțiunea sau comportamentul clasei pe care o poate efectua obiectul unei clase. Are un corp în interiorul parantezelor. În corp, scriem cod care efectuează acțiuni. Poate fi o metodă de instanță sau o metodă statică.
- ▶ **Main method:** O clasă are și metoda principală care oferă punctul de intrare pentru a începe executarea oricărui program.

Variables in Java

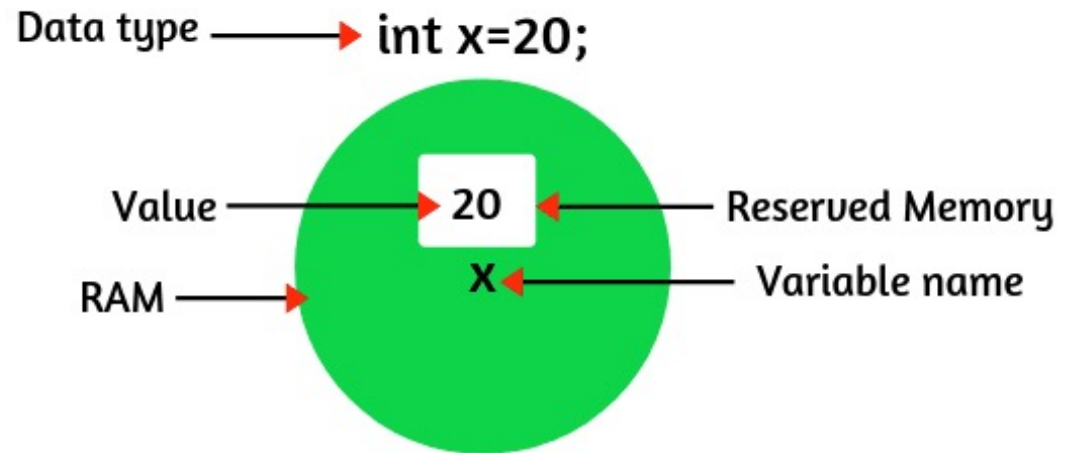
- ▶ O variabilă în Java este un container care tine valoarea atribuita în timpul executării programului Java. Cu alte cuvinte, Variabila este numele locației de memorie rezervată pentru stocarea valorii.
- ▶ Fiecare variabilă din Java are un tip specific de date care determină dimensiunea memoriei. Dimensiunea memoriei rezervate depinde de tipul de date.
- ▶ Există două tipuri de tipuri de date în Java:
 - ▶ Tip de date primitive (stocheaza o singura entitate/valoare)
 - ▶ Tip de date neprimitive (stocheaza un grup de valori)

Variables in Java

- ▶ Știm că avem nevoie de o variabilă pentru a stoca date. Intern, o variabilă reprezintă o locație de memorie în care sunt stocate datele.
- ▶ Când folosim o variabilă în programul java, trebuie să o declarăm mai întâi ca:
- ▶ `int X;`
- ▶ „x” este o variabilă care poate stoca date de tip `int` (numar întreg). Înseamnă că `int` reprezintă natura datelor care pot fi stocate în x. Astfel, `int` se numește `tip de date` în java.

Variables in Java

- ▶ Exemple :
- ▶ **int x = 20;**



A Memory Representation of Variable

Variables in Java

- ▶ În Java, toate variabilele trebuie declarate înainte de a putea fi utilizate în program. Putem declara o variabilă în Java folosind două tipuri de sintaxă

```
data_type variable_name;  
  
data_type variable_name = variable_value;  
|
```

- ▶ Când definim o variabilă cu o anumită valoare inițială, aceasta se numește **inițializare**.

Variables in Java

► Există trei tipuri de variabile în java

1. **Local variables**
2. **Instance variables**
3. **Class/Static variables**

Local Variables in Java

- ▶ O variabilă care este declarată și utilizată în corpul metodelor, constructorilor se numește variabilă locală în Java.
- ▶ Se numește astfel, deoarece variabilele locale nu sunt disponibile pentru utilizare din exterior.
- ▶ Nu au nevoie de acces modifier
- ▶ Variabilelor locale trebuie să li se atribuie o valoare în momentul creării. Dacă utilizați o variabilă locală fără a inițializa o valoare, veți primi o eroare în timpul compilării, cum ar fi „variabila x nu a fost inițializată”.

Instance/Class Variables in Java

- ▶ O variabilă care este declarată în interiorul clasei, dar în afara corpului metodei, constructorului sau a oricărui bloc se numește variabilă de instanță în java.
- ▶ Variabilele de instanță sunt create atunci când un obiect este creat folosind cuvântul cheie „NEW” și sunt distruse atunci când obiectul este distrus.

Static Variables in Java

- ▶ O variabilă declarată cu un cuvânt cheie static se numește variabilă statică în java. O variabilă statică este, de asemenea, numită variabilă de clasă, deoarece este asociată cu clasa.
- ▶ Variabilele statice sunt întotdeauna declarate în interiorul clasei, dar în afara oricăror metode, constructori sau blocuri.

Methods in Java

- ▶ În Java, o metodă este un set de cod utilizat pentru a scrie logica aplicațiilor care efectuează anumite sarcini sau operații specifice.
- ▶ În programarea Java, o metodă este executată atunci când este apelată dintr-o altă metodă.
- ▶ Metoda main () este prima metodă care este executată de JVM (Java Virtual Machine) în programul java.
- ▶ Dacă vrem să executăm orice altă metodă, ar trebui să o numim din metoda main ().
- ▶ Scopul utilizării metodelor în programul Java este de a scrie logica aplicațiilor.

Methods in Java

- Declararea unei metode in Java

```
method_modifier return_type method_name(Parameter_list) {  
    //Method body  
}
```

```
public void calculateSalary(int hourlyRate) {  
    //logic code goes here  
}
```


Getter & Setter Methods

- ▶ O metodă care este utilizată pentru obținerea valorii unei variabile sau returnarea valorii variabilei membre private se numește metoda getter în Java.
- ▶ Această metodă este cunoscută și sub numele de accesori. Pentru fiecare variabilă privată, ar trebui să creăm o metodă getter.
- ▶ Dacă declarăm variabilele de instanță ca fiind private, va trebui să adăugăm metode getter publice pentru fiecare.

Getter & Setter Methods

- ▶ Syntaxa metodelor getter

```
public returnType getPropertyName()
```

- ▶ Daca intoarce o valoare boolean

```
public returnType isPropertyName()
```


Getter & Setter Methods

- ▶ O metodă care este utilizată pentru actualizarea sau setarea valorii unei variabile se numește metoda setter în Java. Această metodă este, de asemenea, cunoscută sub numele de metoda mutator.
- ▶ Folosind metoda setter, putem modifica valoarea unei variabile. La fel ca în cazul metodei getter, ar trebui să creăm o metodă setter pentru fiecare variabilă din clasă.

```
public returnType setPropertyName(data_type Property_value)
```