



# Curs java pentru testare automata

JAVA



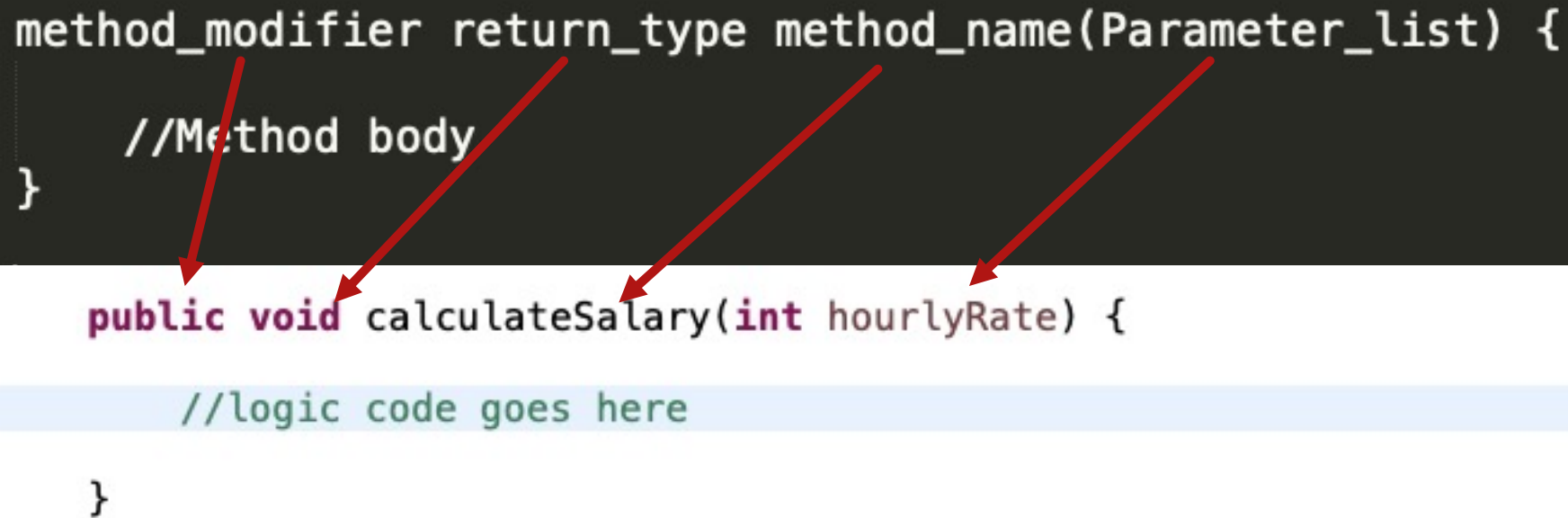
# Methods in Java

- ▶ 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 chemăm din metoda main ().
- ▶ Scopul utilizării metodelor în programul Java este de a scrie logica aplicațiilor și de a reutiliza codul.

# 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  
}
```





# Methods in Java

- ▶ Numele metodei cu lista parametrilor (numărul de parametri, tipul parametrilor și ordinea parametrilor) se numește **semnătura metodei** în java.
- ▶ **Lista parametrilor**: Lista parametrilor este o listă separată prin virgulă cu zero sau mai multe declarații de parametri.
- ▶ Sintaxă: **parameter-modifier data-type parameter-name**
- ▶ **parameter-modifier** poate fi final.
- ▶ Dacă parametrul este final, nu poate fi modificat în interiorul metodei.
- ▶ Tipul de date poate fi int, float, char, double, etc. Fiecare parametru trebuie să fie un nume distinct.



# Methods in Java

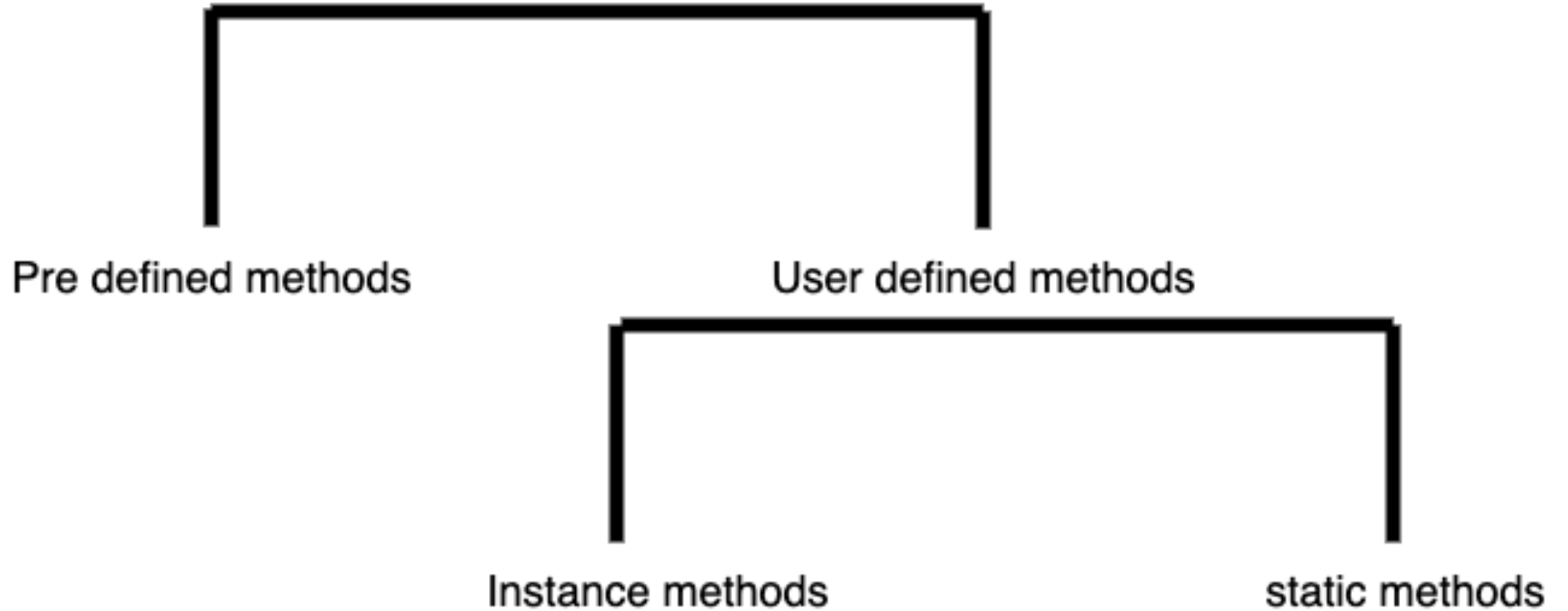
- ▶ Dacă există mai mulți parametri, aceștia sunt separați prin virgule.
- ▶ Dacă nu există parametri, includeți o pereche goală de paranteze ().
- ▶ Lista parametrilor metodei face posibilă transmiterea valorii metodei.
- ▶ Este declarat în paranteze după numele metodei.
- ▶ Domeniul de aplicare al listei de parametri se află în corpul metodei.
- ▶ Metoda poate conține, de asemenea, zero parametri.



# Methods in Java

- ▶ Corpul metodei: corpul metodei este un bloc de cod care conține instrucțiuni, precum și declarația variabilelor locale și a claselor locale.
- ▶ Corpul metodei poate conține, de asemenea, declarații de returnare.
- ▶ Dacă tipul de returnare este null, metoda poate efectua o sarcină fără a returna nicio valoare.
- ▶ O astfel de metodă se numește metodă void. Nu va exista nicio declarație de returnare în corpul metodei.

## Methods in java





# Pre defined Methods in Java

- ▶ Metodele predefinite în Java sunt acele metode care sunt deja definite în Java API (Application Programming Interface) pentru a fi utilizate într-o aplicație.
- ▶ Limbajul de programare Java conține clase predefinite care sunt organizate în diferite pachete predefinite.
- ▶ În cadrul acestor clase predefinite, există metode predefinite.
- ▶ De fapt, Java are peste 160 de pachete predefinite care conțin peste 3000 de clase predefinite.
- ▶ Aceste clase au mult mai multe mii de metode individuale predefinite.
- ▶ Toate sunt disponibile programatorilor prin intermediul API-ului Java.



# User defined Methods in Java

- ▶ Metodele definite de utilizator în Java sunt acele metode care sunt definite în interiorul unei clase pentru a efectua o sarcină sau o funcție specială într-o aplicație.
- ▶ Putem declara două tipuri de metode definite de utilizator într-o aplicație.
- ▶ 1. Metoda instanței
- ▶ 2. Metoda statică



# Instance Methods in Java

- ▶ O metodă de instanță este utilizată pentru a implementa comportamentele fiecărui obiect / instanță din clasă.
- ▶ Deoarece metoda instanței reprezintă comportamente ale obiectelor.
- ▶ Prin urmare, metoda instanței este legată de un obiect.
- ▶ O metodă de instanță este, de asemenea, cunoscută sub numele de metodă nestatică.
- ▶ Deci, fără a crea un obiect al clasei, metodele nu pot exista pentru a-și îndeplini comportamentele sau sarcinile dorite.
- ▶ Este alocată în memoria heap în timpul creării obiectului.



# Static Methods in Java

- ▶ Când declarați orice metodă cu un modifier static, aceasta se numește metodă statică în java.
- ▶ O metodă statică este legată de clasă.
- ▶ Prin urmare, este cunoscută și ca metodă de clasă.
- ▶ Este folosită pentru a implementa comportamentul clasei în sine.
- ▶ Metodele statice se încarcă în memorie în timpul încărcării clasei și înainte de crearea obiectului.
- ▶ Note:
  - ▶ 1. O metodă de instanță se poate referi la variabilele statice (variabile de clasă), precum și la variabilele de instanță ale clasei.
  - ▶ 2. O metodă statică se poate referi doar la variabile statice.



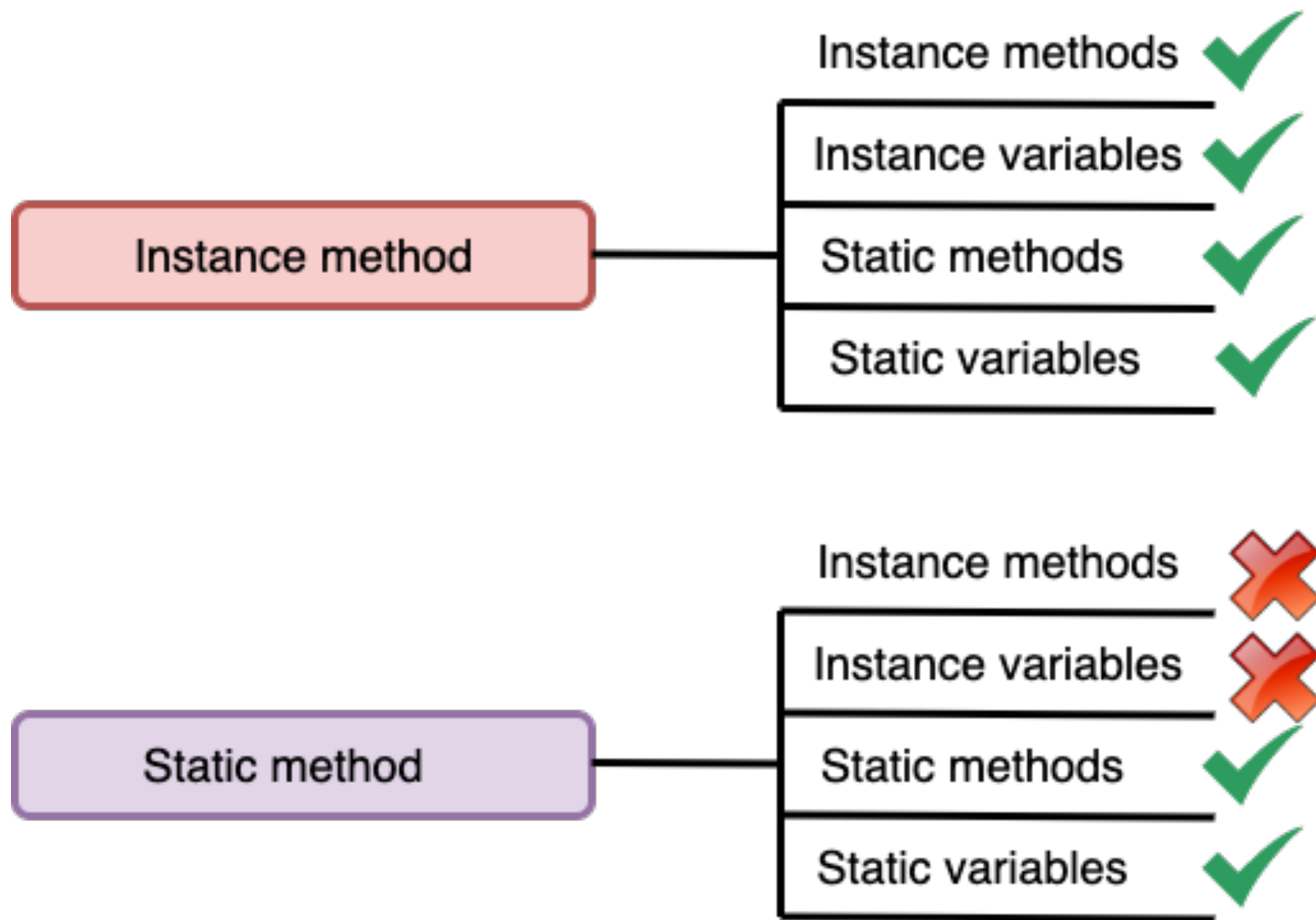
# Static vs non static calls in Java

- ▶ Nu putem accesa direct membri nestatici dintr-o metodă statică. Adică, nu putem apela direct membrii instanței într-o zonă statică, deoarece există un concept logic în spatele acestei restricții.
- ▶ Membrii statici precum variabilele statice, metodele sau blocurile sunt legate de clasă.
- ▶ Prin urmare, primesc memorie o singură dată când clasa este încărcată în memorie.
- ▶ Dar membrii nestatici, cum ar fi variabilele de instanță, metoda sau blocul, obțin memorie după crearea obiectului clasei.



# Static vs non static calls in Java

- ▶ Prin urmare, dacă apelăm membrii non-statici din zona statică, înseamnă că atunci când clasa este încărcată în memorie, membrii statici sunt încărcati și va căuta membri nestatici care nu există deoarece nu avem creată o instanță până acum.
- ▶ Prin urmare, există ambiguitate.
- ▶ De aceea nu putem apela membrul nestatic din zona statică.
- ▶ O metodă nestatică poate accesa membrii statici în Java
- ▶ Deoarece metoda statică nu poate accesa direct membrii non-statici, dar dacă creăm obiectul clasei în cadrul metodei statice, în acest caz, putem accesa membrii non-statici în cadrul metodei statice.





## Parameters /arguments

- ▶ **Argumentele** în Java sunt valorile reale care sunt transmise variabilelor definite în antetul metodei atunci când metoda este apelată dintr-o altă metodă.
- ▶ Adică, ori de câte ori este apelată o anumită metodă în timpul executării programului, există câteva valori care sunt trecute pentru a apela acea metodă specială. Aceste valori se numesc argumente.
- ▶ Valorile argumentelor trecute înlocuiesc acei parametri care au fost utilizați în timpul definirii metodei și corpul metodei este apoi executat cu aceste valori.
- ▶ Tipul valorilor argumentelor transmise trebuie să fie același cu tipul specificat pentru parametrul corespunzător în definiția metodei.
- ▶ Uneori, un argument este numit și parametru real.



## Parameters /arguments

- ▶ În Java, un **parametru** este un nume de variabilă cu tip care este declarat în semnătura metodei. Lista parametrilor este inclusă între paranteze.
- ▶ Fiecare parametru este format din două părți:
- ▶ tipul și numele variabilei.
- ▶ Un tip de date urmat de un nume de variabilă definește tipul de valoare care poate fi transmis unei metode atunci când este apelată.
- ▶ Se mai numește adesea parametru formal.
- ▶ Parametrii declarați în semnătura metodei sunt întotdeauna variabile locale care primesc valori atunci când metoda este apelată dintr-o altă metodă.



## Return type

- ▶ O metodă poate accepta date din exterior și poate, de asemenea, să returneze rezultatele.
- ▶ Pentru a returna rezultatul, o instrucțiune **return** este utilizată în interiorul unei metode pentru a ieși din aceasta la metoda apelantă.
- ▶ În Java, returnarea este un cuvânt cheie care este folosit pentru a ieși din metodă numai cu sau fără a returna o valoare.
- ▶



# Return type

- ▶ Tipul de returnare poate fi un tip de date primitiv, cum ar fi int, float, double, un tip de referință sau un tip void care reprezintă „returnează nimic”.
- ▶ Adică nu dau nimic înapoi. Când se apelează o metodă, metoda poate returna o valoare metodei apelantului.




```
Shape obj = new Shape();  
System.out.println("Aria patratului este :" + obj.calculateSquareArea( lenght: 4));
```

```
public int calculateSquareArea(int lenght){  
    int squareArea = lenght * lenght;  
    return squareArea;  
}  
{
```

```
Aria patratului este :16
```

```
Shape obj = new Shape();  
System.out.println("Aria patratului este :" + obj.calculateSquareArea( length: 4));
```

```
public int calculateSquareArea(int length){  
    int squareArea = length * length;  
    return squareArea;  
}
```

 Parametru

```
Aria patratului este :16
```



```
Shape obj = new Shape();
```

```
System.out.println("Aria patratului este :" + obj.calculateSquareArea( lenght: 4));
```

Argument



```
public int calculateSquareArea(int lenght){  
    int squareArea = lenght * lenght;  
    return squareArea;  
}
```

Parametru



```
Aria patratului este :16
```

```
Shape obj = new Shape();
```

```
System.out.println("Aria patratului este :" + obj.calculateSquareArea( lenght: 4));
```

Argument

Valoarea aceasta este folosita pentru lenght

```
public int calculateSquareArea(int lenght){  
    int squareArea = lenght * lenght;  
    return squareArea;  
}
```

Parametru

```
Aria patratului este :16
```



```
Shape obj = new Shape();
```

```
System.out.println("Aria patratului este :" + obj.calculateSquareArea( lenght: 4));
```

Argument

Valoarea aceasta este folosita pentru lenght

```
public int calculateSquareArea(int lenght){
```

```
    int squareArea = lenght * lenght;
```

```
    return squareArea;
```

```
}
```

Parametru

Reprezinta valoarea pe care variabila o  
va avea, in cazul acesta  $4 \times 4 = 16$

```
Aria patratului este :16
```

```
Shape obj = new Shape();
```

```
System.out.println("Aria patratului este :" + obj.calculateSquareArea( lenght: 4));
```

Argument

Valoarea aceasta este folosita pentru lenght

```
public int calculateSquareArea(int lenght){
```

```
    int squareArea = lenght * lenght;
```

```
    return squareArea;
```

```
}
```

Parametru

Reprezinta valoarea pe care variabila o  
va avea, in cazul acesta  $4 \times 4 = 16$

Variabila acesta exista  
doar cat metoda este executata

```
Aria patratului este :16
```



# 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)

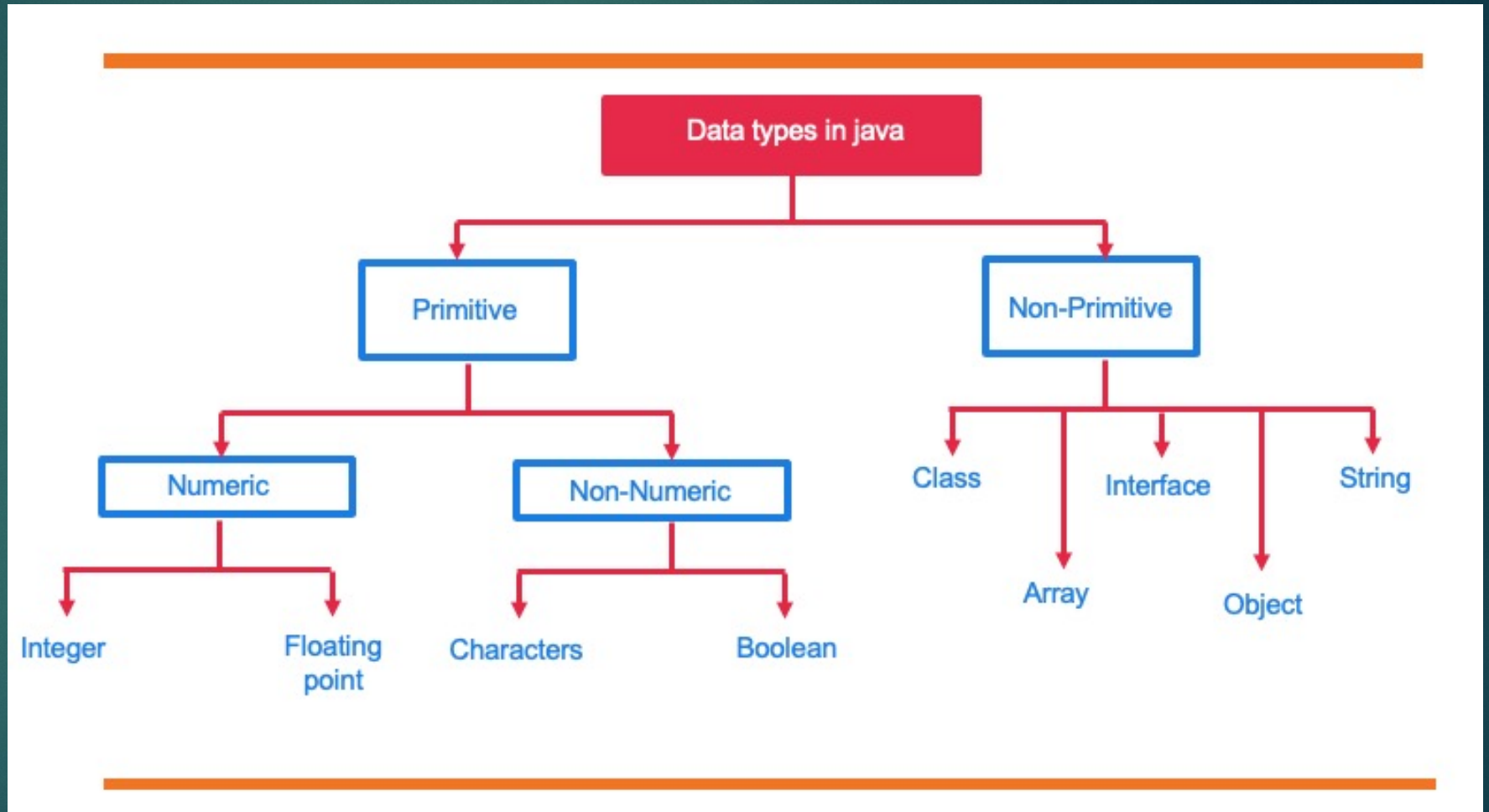


# Variables in Java

- ▶ 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.

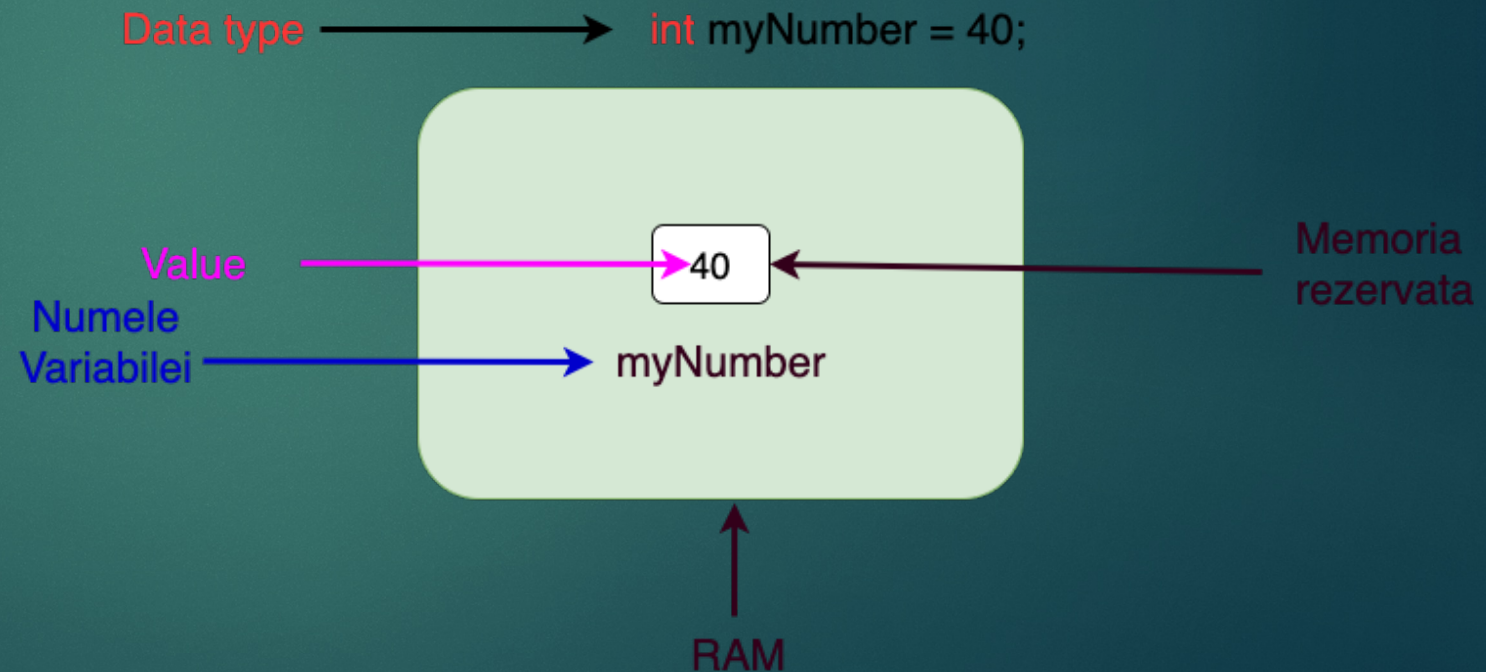


# Variables in Java



# Variables in Java

- ▶ Exemplu :
- ▶ `int myNumber = 40;`





# 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.



# Constants in Java

- ▶ O variabila care are o valoare care este fixă și nu se modifică în timpul executării unui program se numește constanta în java.
- ▶ Cu alte cuvinte, constantele Java sunt valori fixe care nu pot fi modificate (le declaram ca si final)
- ▶ Java acceptă diferite tipuri de constante:
  - Integer Constants
  - Real Constants
  - Character Constants
  - String Constants

```
//o variabila a carei valoare nu o schimbam  
public static final int SPEED_OF_LIGHT = 186000;  
|
```



# Data Types

- ▶ 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;`
- ▶ Aici, „x” este o variabilă care poate stoca date de tip `int` (nr întreg). Înseamnă că `int` reprezintă natura datelor care pot fi stocate în x. Astfel, `int` se numește tip de date (data type) în java.



# Data types

- ▶ Limbajele de programare sunt fie dynamically typed fie statically typed.
- ▶ Dynamically typed : determină tipul de date al unei variabile în timpul rulării - adică în timp ce programul rulează. Prin urmare, în aceste tipuri de limbaje programatorul nu trebuie să specifice tipul de date al unei variabile.
- ▶ Java este un limbaj de programare statically typed - ceea ce înseamnă că se așteaptă ca variabilele sale să fie declarate înainte ca acestea să poată fi atribuite valori deoarece, în limbajele statically typed, tipul de date este verificat în momentul compilării.



# Data types

- ▶ În timp ce Java este un limbaj statically typed , în versiunea 10 a Java au adăugat suport pentru type inference - ceea ce înseamnă că putem declara o variabilă ca var și Java va deduce tipul de date pe baza a ceea ce este atribuit variabilei.



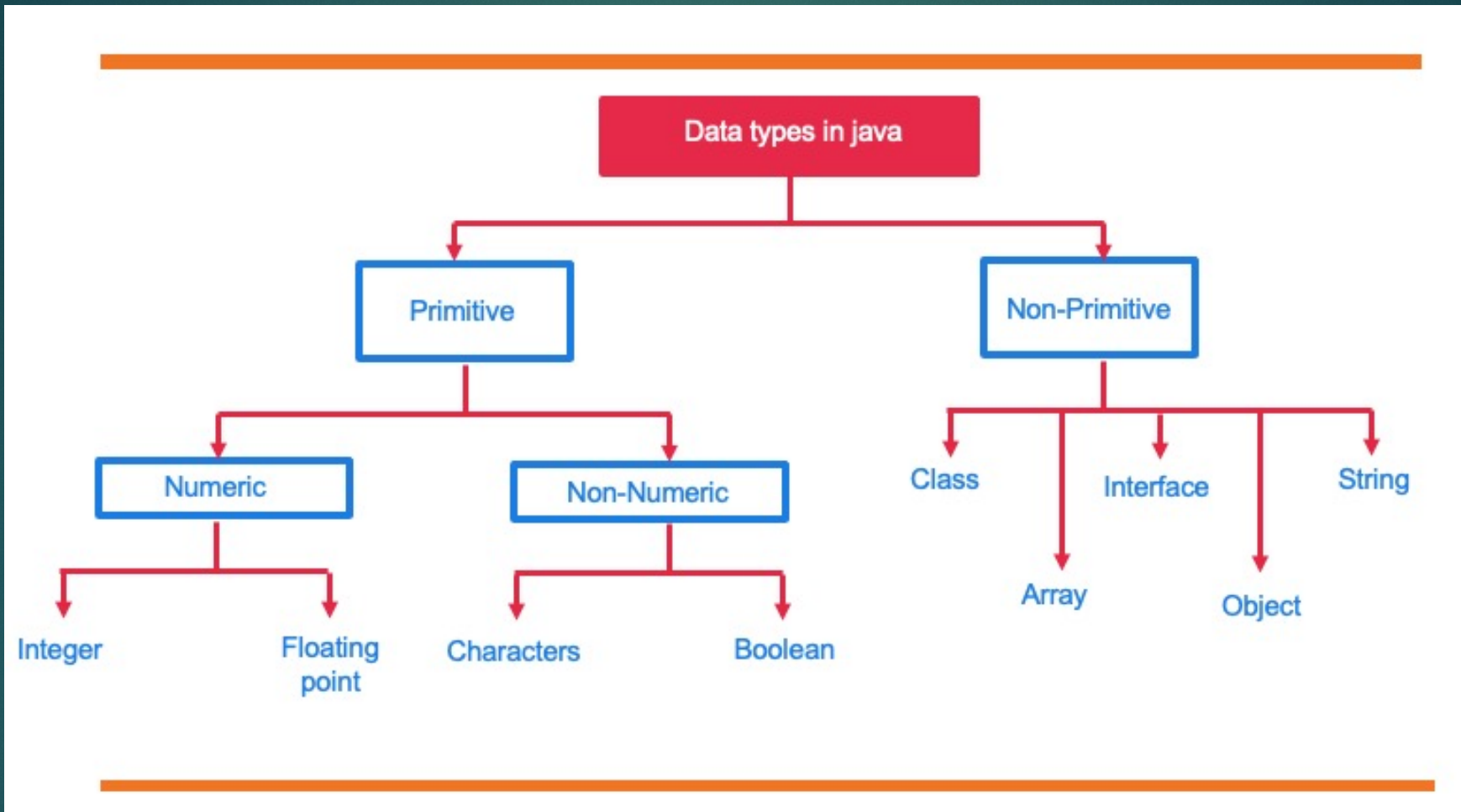
# Data Types

- ▶ Data Type în java este un termen care specifică dimensiunea memoriei și tipul de valori care pot fi stocate în locația memoriei.
- ▶ Cu alte cuvinte, tipurile de date definesc diferite valori pe care le poate lua o variabilă.
- ▶ **Exemplu:**
- ▶ 1. Variabila x poate stoca un număr întreg ca 100 ca:
- ▶ `Int x = 100; // Aici = reprezintă că valoarea 100 este stocată în x.`



# Data Types

- ▶ In Java oferă tipurile de date sunt împărțite în două categorii după cum urmează:
- ▶ **Primitive data types:** Tipuri de date primitive (numite și tipuri intrinseci sau încorporate)
- ▶ **Non-primitive data types:** Tipuri de date neprimitive (numite și tipuri de date derivate sau de referință)





## Integer data types

```
graph TD; A[Integer data types] --> B[Byte]; A --> C[Short]; A --> D[Int]; A --> E[Long];
```

Byte

Short

Int

Long

## Integer data types

Byte

1 byte

Short

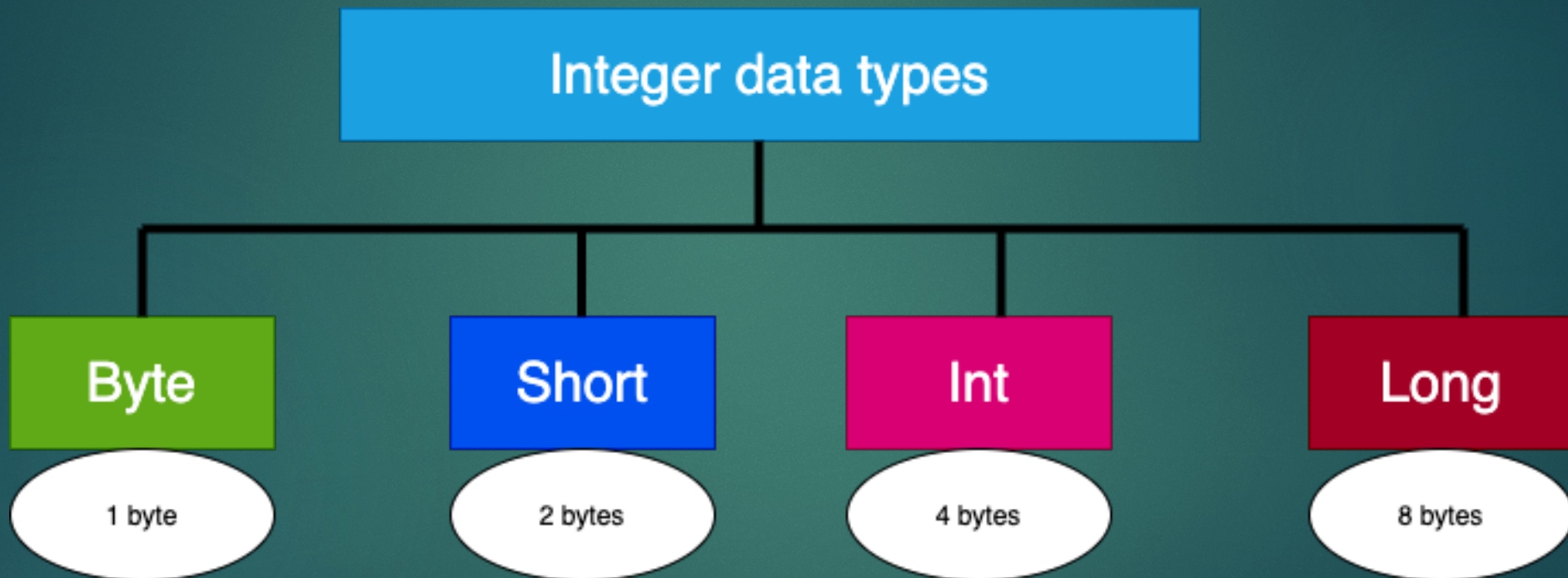
2 bytes

Int

4 bytes

Long

8 bytes





## Floating data types

```
graph TD; A[Floating data types] --> B[float]; A --> C[double]; B --> D(4 byte); C --> E(8 bytes)
```

float

4 byte

double

8 bytes



# Primitive data types

- ▶ Tipurile de date primitive în Java sunt acele tipuri de date ale căror variabile pot stoca o singură valoare la un moment dat.
- ▶ Nu putem stoca mai multe valori de același tip. Aceste tipuri de date sunt predefinite în Java.
- ▶ Tipurile de date primitive nu sunt tipuri de date definite de utilizator.
- ▶ Adică programatorii nu pot dezvolta tipuri de date primitive.



# Primitive data types

- ▶ Java definește opt tipuri de date primitive:
  - ▶ boolean
  - ▶ char
  - ▶ Byte
  - ▶ Short
  - ▶ Int
  - ▶ Long
  - ▶ Char
  - ▶ Float
  - ▶ double
- ▶ Acestea pot fi clasificate în continuare în patru grupuri.



# Primitive data types

- ▶ 1. Tip de date logice : boolean
- ▶ 2. Tip de date pentru categoria de caractere: char
- ▶ 3. Tipuri de date ale categoriei numerelor întregi: byte, short, int și long.
- ▶ 4. Tipuri de date de tip real: float și double



Data type	Valori	Valoare implicita	Dimensiune
byte	-128 → 127	0	1 byte
short	-32768 → 32767	0	2 bytes
int	-2147483648 → 2147483647	0	4 bytes
long	-9,223,372,036,854,775,808 → 9,223,372,036,854,775,807	0L	8 bytes
float	7 cifre semnificative	0.0f	4 bytes
double	15 cifre semnificative	0.0d	8 bytes
char	'\u0000' → '\uffff' 0 → 65535	'\u0000'	2 bytes
boolean	true sau false	false	1 bit

# Byte data type

- ▶ Un byte este cel mai mic tip de date întregi care are cea mai mică dimensiune de memorie alocată și este utilizat în cea mai mare parte pentru a economisi memorie.
- ▶ Dimensiunea memoriei de tip byte(octet) este de 8 biți (adică 1 octet). Un octet este egal cu 8 biți.
- ▶ Un byte reprezintă de la 0 la 127 pe partea pozitivă și de la -1 la -128 pe partea negativă. Deci poate reprezenta totalul de 256 ( $2^8$ ) numere.
- ▶ Valoarea implicită a octetului este 0.



# Short data type

- ▶ Tipul de date short are o dimensiune de memorie mai mare decât byte și mai mică decât Integer.
- ▶ Dimensiunea de memorie implicită este alocată 16 biți, adică 4 octeți.
- ▶ Valoarea implicită este 0.



# Int data type

- ▶ Acest tip de date este utilizat pentru valori întregi în programarea Java.
- ▶ Dimensiunea memoriei este de 32 de biți, adică 4 octeți, iar valoarea implicită este 0.



# Long data type

- ▶ Acest tip de date este utilizat mai ales pentru un număr mare în care tipul `int` nu este suficient de mare pentru a stoca valoarea dorită.
- ▶ Dimensiunea implicită a memoriei alocată acestui tip de date este de 64 de biți, adică 8 octeți(bytes), iar valoarea implicită este 0.
- ▶ Tipul de date `long` este util atunci când sunt necesare numere întregi mari.
- ▶ Exemplu: `long num = -2334456L;`
- ▶ `L` reprezintă că JVM o va considera ca fiind o valoare `long` și îi va alocă 8 octeți.



# Double data type

- ▶ Un tip de date `double()` este utilizat pentru a reprezenta cu exactitate numerele zecimale de până la 15 cifre zecimale.
- ▶ Dimensiunea memoriei este de 64 de biți, adică 8 octeți, iar valoarea implicită este 0,0d.
- ▶ Exemplu: `double num = 1345,6;`

`double num2 = 1,50e9; // Aici, e reprezentă  $\times 10$  la putere.`

Prin urmare, `1.50e9` înseamnă  $1.50 \times 10^9$ . Se numește notăție științifică (exponențială) a numărului reprezentativ.



# Float data type

- ▶ Un tip de date float este utilizat pentru a reprezenta numărul zecimal care poate conține 6 până la 7 cifre zecimale.
- ▶ Dimensiunea de memorie implicită alocată pentru acest tip de date este de 32 de biți, adică 4 octeți și valoarea implicită este 0,0f.
- ▶ Exemplu: float num = 10.6f;
- ▶ Dacă nu scriem „f”, atunci JVM va considera ca fiind dublu și ar fi alocat 8 octeți. În acest caz, va apărea o eroare „Type mismatch: cannot convert from double to float”.
- ▶ Dacă folosim f, JVM o va considera ca valoare float și va alocă doar 4 octeți(bytes).



# Boolean data type

- ▶ Tipul de date boolean reprezintă un bit de informații fie ca fiind adevărat sau fals.
- ▶ Adică există doar două valori posibile adevărat sau fals.
- ▶ Intern, JVM folosește un bit de stocare pentru a reprezenta o valoare booleană.
- ▶ Este, în general, utilizat pentru a testa o anumită declarație condițională în timpul executării programului.
- ▶ Valoarea implicită este falsă (false)
- ▶ Exemplu : `boolean b = false;`



# Char data type

- ▶ Un tip de date char este utilizat în principal pentru a stoca un singur caracter precum P, a, b, z, x etc.
- ▶ Este un singur caracter Unicode pe 16 biți.
- ▶ Dimensiunea memoriei luate de un singur caracter este de 2 octeți.
- ▶ Poate reprezenta un interval de la 0 la 65536 caractere.
- ▶ O valoare implicită pentru caracter este „u0000”, care reprezintă spațiu gol sau spațiu unic. Aici, „u” reprezintă faptul că caracterul este un Unicode.
- ▶ Exemplu : `char ch = 'D';`



# Non - Primitive data types (Referinta)

- ▶ Tipurile de date Referinta sunt create de programatori.
- ▶ Ele nu sunt predefinite în java ca tipurile de date primitive. Aceste tipuri de date sunt utilizate pentru a stoca un grup de valori sau mai multe valori.
- ▶ De exemplu, luăm o matrice. Poate stoca un grup de valori. În mod similar, un alt exemplu este o clasă care poate stoca valori diferite. Prin urmare, aceste tipuri de date sunt cunoscute și sub numele de tipuri avansate de date (advanced data types) în Java.



# Non - Primitive data types (Referinta)

- ▶ Când definim o variabilă de tipuri de date Referinta, aceasta face referire la o locație de memorie în care datele sunt stocate în memoria heap. Adică se referă la o memorie în care este plasat efectiv un obiect.
- ▶ Astfel, variabila unui tip de date non-primitiv este numită și tip de date de referință în Java sau pur și simplu variabilă de referință de obiect.
- ▶ În programarea Java, toate tipurile de date non-primitive sunt pur și simplu numite obiecte care sunt create prin instanțierea unei clase.



# Non - Primitive data types (Referinta)

- ▶ Există cinci tipuri de tipuri de date neprimitive în Java
- ▶ 1. Class
- ▶ 2. Object
- ▶ Clasă și obiecte: Fiecare clasă este tip de date și este considerată, de asemenea, ca tip de date definite de utilizator. Acest lucru se datorează faptului că un utilizator creează o clasă.



# Non - Primitive data types (Referinta)

- ▶ 3. String
- ▶ Un String (șir) reprezintă o secvență de caractere precum : ABC123.
- ▶ Cel mai simplu mod de a crea un obiect de tip String este stocarea secvenței de caractere într-o variabilă de tip String astfel:
  - ▶ `String str = „orice”;` Aici, variabila de tip string conține „orice”. Un String este, de asemenea, o clasă.



# Non - Primitive data types (Referinta)

- ▶ 4. Array
- ▶ Un Array în java este un obiect care este utilizat pentru a stoca mai multe variabile de același tip.
- ▶ Aceste variabile pot fi tip de date primitive sau neprimitive.
- ▶ Exemplul declarării unei variabile Array de tip primitiv de date int este următorul:
- ▶ `int [] note;`
- ▶ Exemplul declarării unei variabile matrice de tip non-primitiv de date este :
- ▶ `Tester [] testers; // Testser este un nume de clasă.`



# Non - Primitive data types (Referinta)

- ▶ 5. Interface
- ▶ O interfață este declarată ca o clasă, dar singura diferență este că ea conține doar variabile finale și declarații de metodă.
- ▶ Este o clasă complet abstractă.



# Array

- ▶ Array sunt obiecte care stochează mai multe variabile de același tip.
- ▶ Poate conține tipuri primitive, precum și referințe la obiecte (non primitive).
- ▶ Deoarece Array sunt obiecte, acestea sunt create în timpul rulării.
- ▶ Lungimea Array -ului este fixă.



# IF else

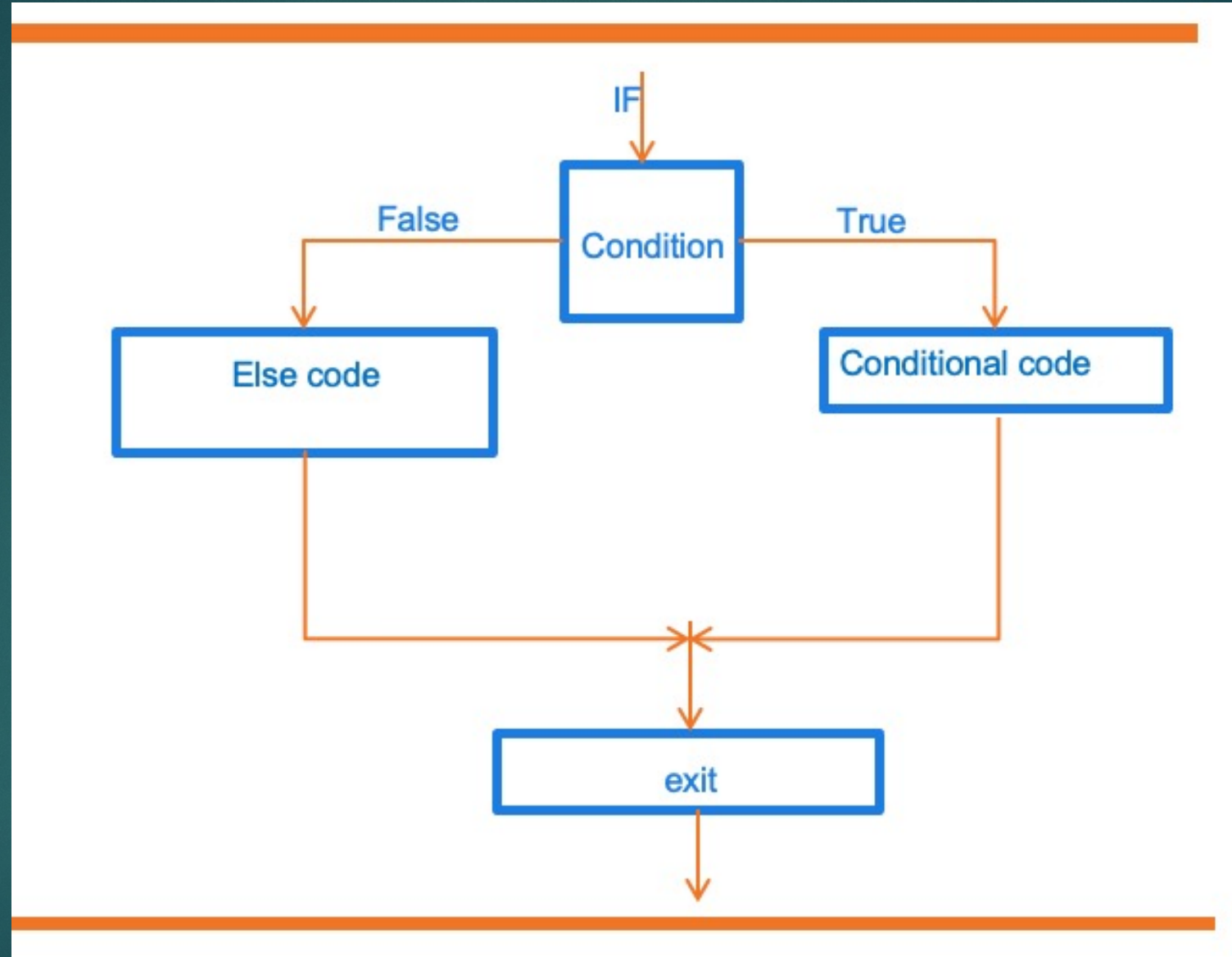
- ▶ Un if else în Java este o declarație condițională bidirecțională care decide calea de execuție în funcție de condiția adevărată sau falsă.
- ▶ Cu alte cuvinte, declarația if-else este utilizată pentru a efectua o acțiune specifică, în funcție de faptul dacă o condiție specificată este adevărată sau falsă.
- ▶ Aici, o acțiune semnifică o afirmație sau un grup de afirmații.



# IF else

- ▶ O instrucțiune unidirecțională execută o instrucțiune dacă și numai dacă condiția specificată este adevărată. Dacă condiția este falsă, nu se va face nimic.
- ▶ Dar să presupunem că vrem să luăm acțiuni alternative atunci când condiția specificată este falsă. În acest caz, vom folosi o instrucțiune bidirecțională if-else.
- ▶ O instrucțiune bidirecțională if-else în Java este utilizată pentru a direcționa executarea programului prin două căi diferite, în funcție de condiția adevărată sau falsă.





```
int number = 5;

if (number % 2 == 0) {

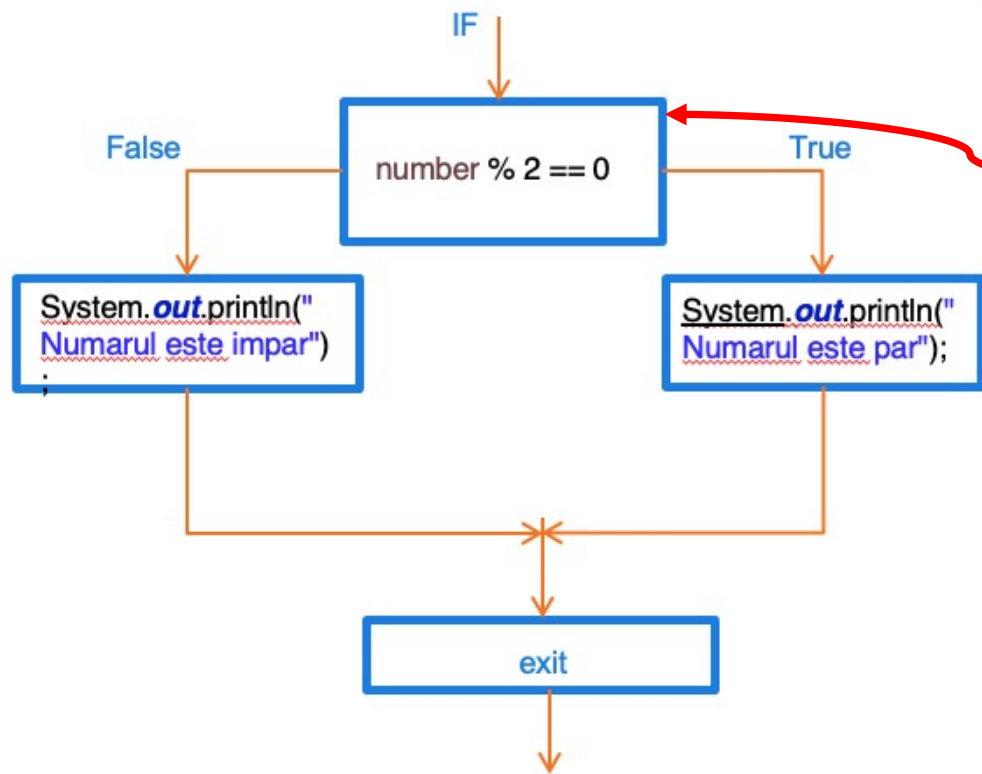
    //Code executed if boolean expresion
    // or condition is true.
    System.out.println("Numarul este par");

}
else {

    //Code executed if boolean expresion
    // or condition is false.
    System.out.println("Numarul este impar");
}
```

# IF else





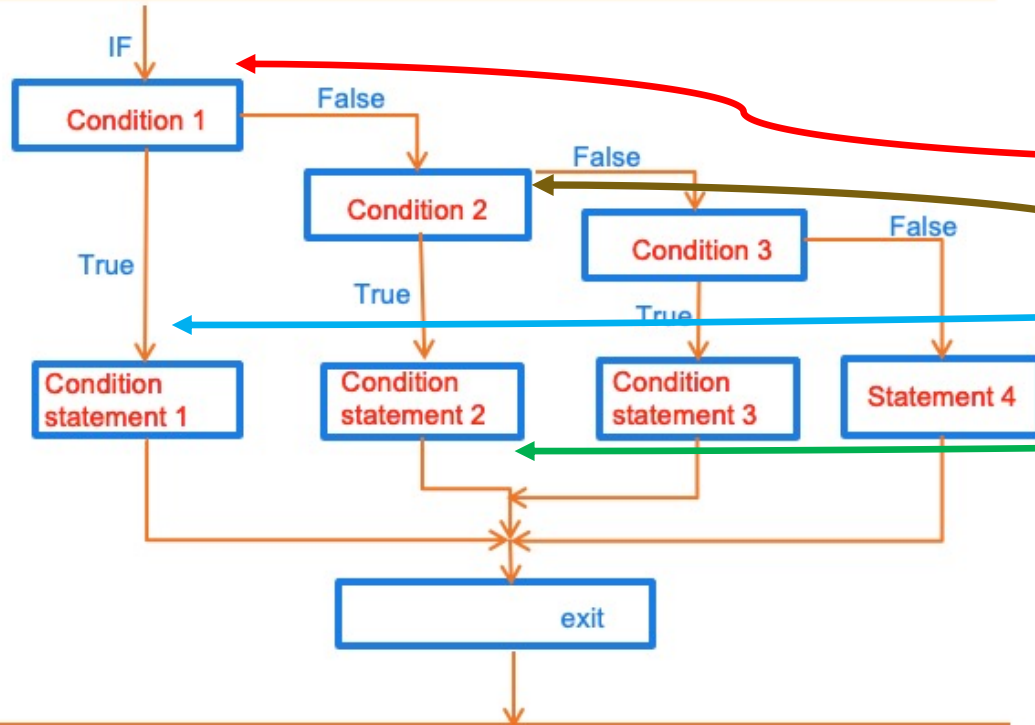
```
int number = 5;
if (number % 2 == 0) {
    //Code executed if boolean expression
    // or condition is true.
    System.out.println("Numarul este par");
}
else {
    //Code executed if boolean expression
    // or condition is false.
    System.out.println("Numarul este impar");
}
```

# IF ELSE-if

- ▶ if-else if în Java este o structură de decizie multi-way care este utilizată pentru a decide între trei sau mai multe acțiuni.

```
if(condition_1) {  
    //Conditional statement 1;  
}  
else if(condition_2) {  
    //Conditional statement 2;  
}  
else if(condition_3) {  
    //Conditional statement 3;  
}  
else {  
    //Statement 4;  
}
```





```
if(condition_1) {  
    //Conditional statement 1;  
}  
else if(condition_2) {  
    //Conditional statement 2;  
}  
else if(condition_3) {  
    //Conditional statement 3;  
}  
else {  
    //Statement 4;  
}
```

# IF ELSE-IF

- ▶ if-else if funcționează în următorii pași :
- ▶ 1. Prima condiție specificată este evaluată ca fiind adevărată. Dacă condiția este adevărată, instrucțiunea 1 este executată și restul părții este ocolită.
- ▶ 2. Dacă condiția 1 este falsă, a doua condiție este evaluată ca fiind adevărată. Dacă cea de-a doua condiție este adevărată, instrucțiunea2 este executată, iar restul părții este ocolită.
- ▶ 3. Dacă a doua condiție este falsă, a treia condiție și restul condițiilor (dacă este necesar) sunt evaluate (sau testate) până când se îndeplinește o condiție sau toate condițiile se dovedesc a fi false.



# IF ELSE-IF

- ▶ 4. Dacă toate condițiile sunt false, instrucțiunea finală else (adică declarația 4 în exemplul de mai sus) va fi executată.
- ▶ Dacă nu există nicio afirmație finală și niciuna dintre condiții nu este adevărată, nu va avea loc nicio acțiune.
- ▶ O condiție este evaluată numai atunci când toate condițiile care vin înainte de aceasta sunt false.

```
Shape obj = new Shape();
```

```
System.out.println("Aria patratului este :" + obj.calculateSquareArea( lenght: 4));
```

Argument

Valoarea aceasta este folosita pentru lenght

```
public int calculateSquareArea(int lenght){  
    int squareArea = lenght * lenght;  
    return squareArea;  
}
```

Parametru

Reprezinta valoarea pe care variabila o va avea, in cazul acesta  $4 \times 4 = 16$

Variabila acesta exista doar cat metoda este executata

Aria patratului este :16

Reprezinta ceea ce metoda intoarce, in cazul nsotru valoarea 16