Intro Java

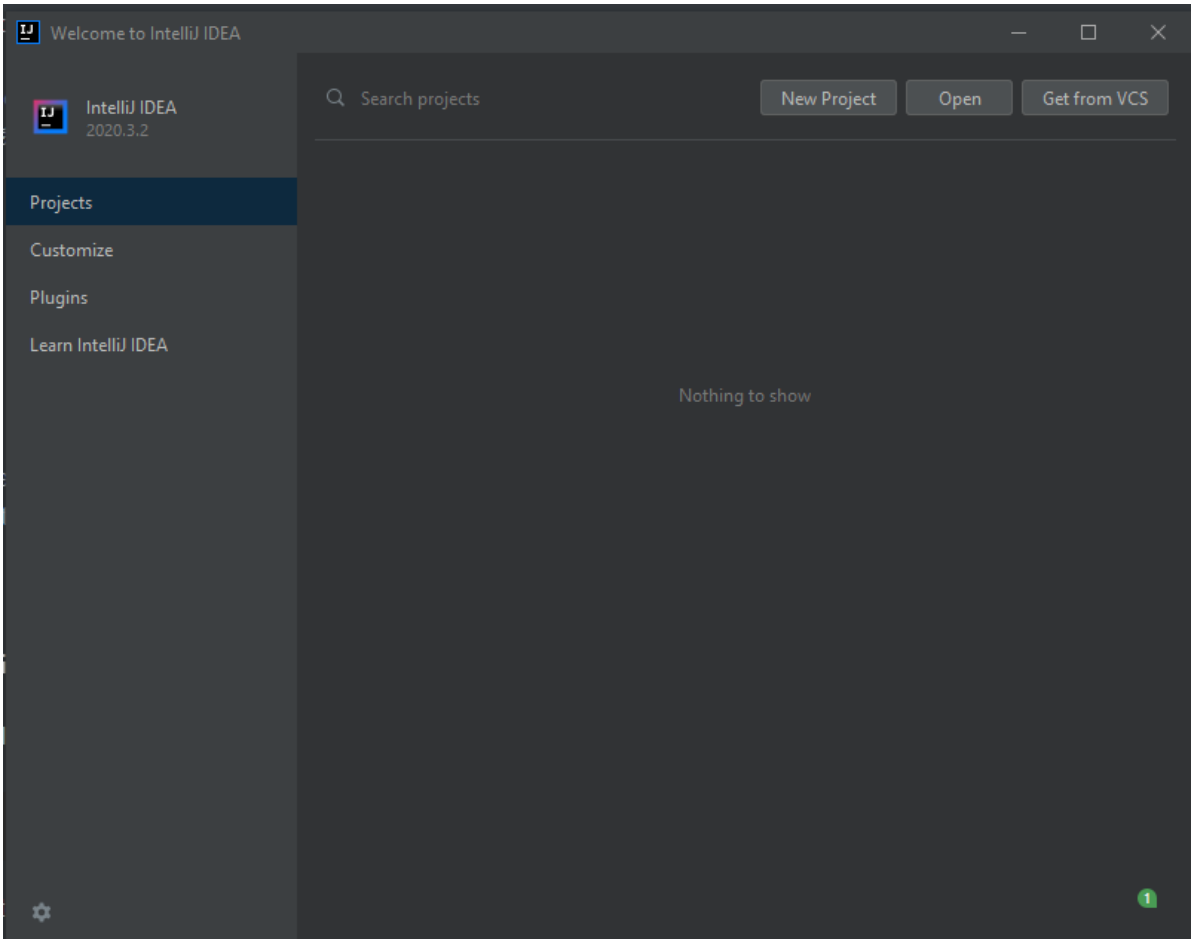
Versiune	5
Data	21-Apr-2023

Cum deschidem un proiect nou in IntelliJ IDEA

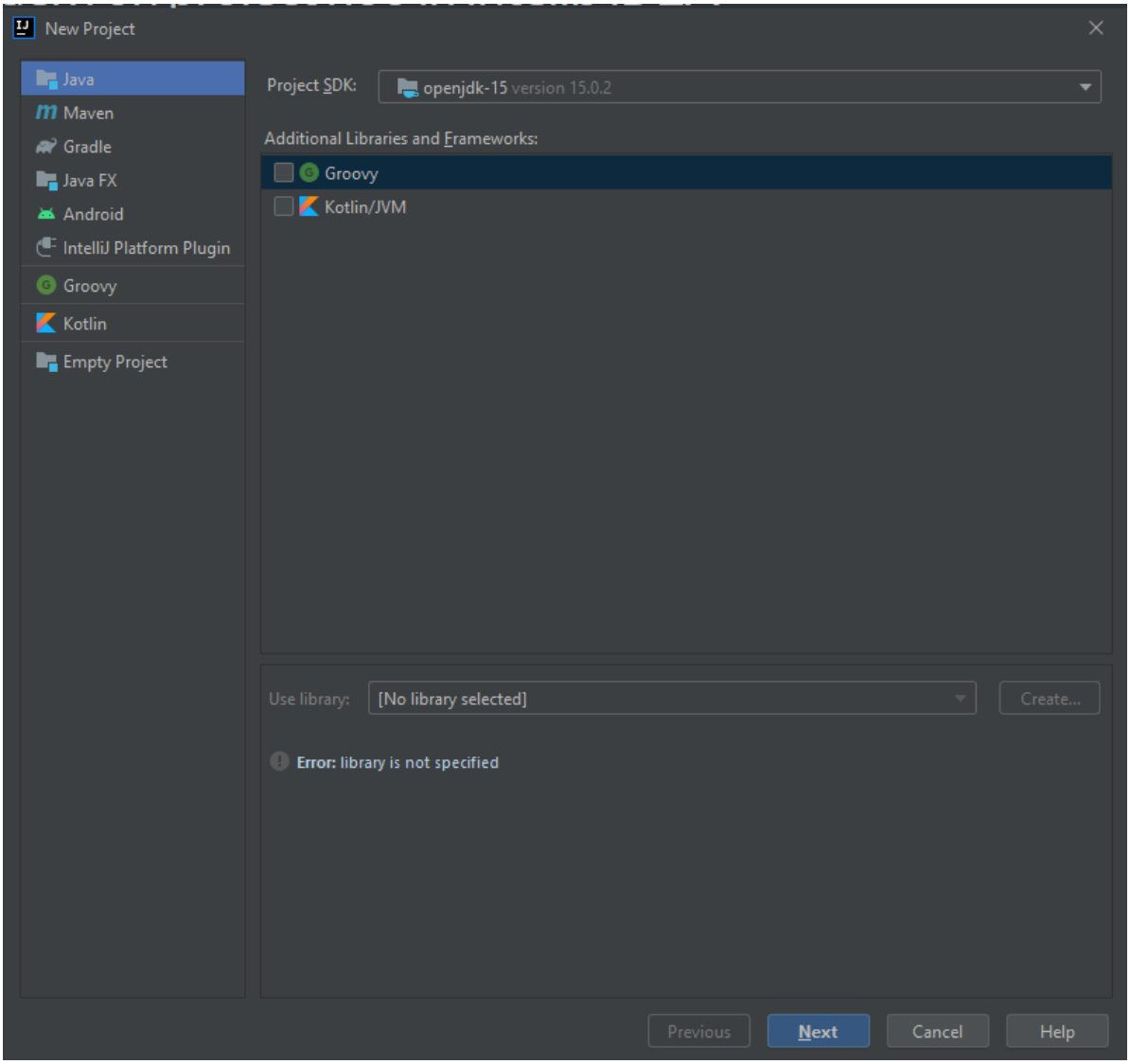
IntelliJ IDEA este un mediu de dezvoltare integrat (IDE) scris în Java pentru dezvoltarea de programe software. Este dezvoltat de JetBrains (cunoscut anterior ca IntelliJ) și este disponibil ca o ediție comunitară (community) și într-o ediție comercială. Ambele pot fi folosite pentru dezvoltare comercială.

Pentru a crea un nou proiect, deschideți IntelliJ și ar trebui să se deschidă o fereastră similară cu aceasta. Dacă IntelliJ deschide proiectul curent, mergeți mai jos la secțiunea în care vă prezint ce trebuie să faceți în cazul în care este deschis un proiect deja existent

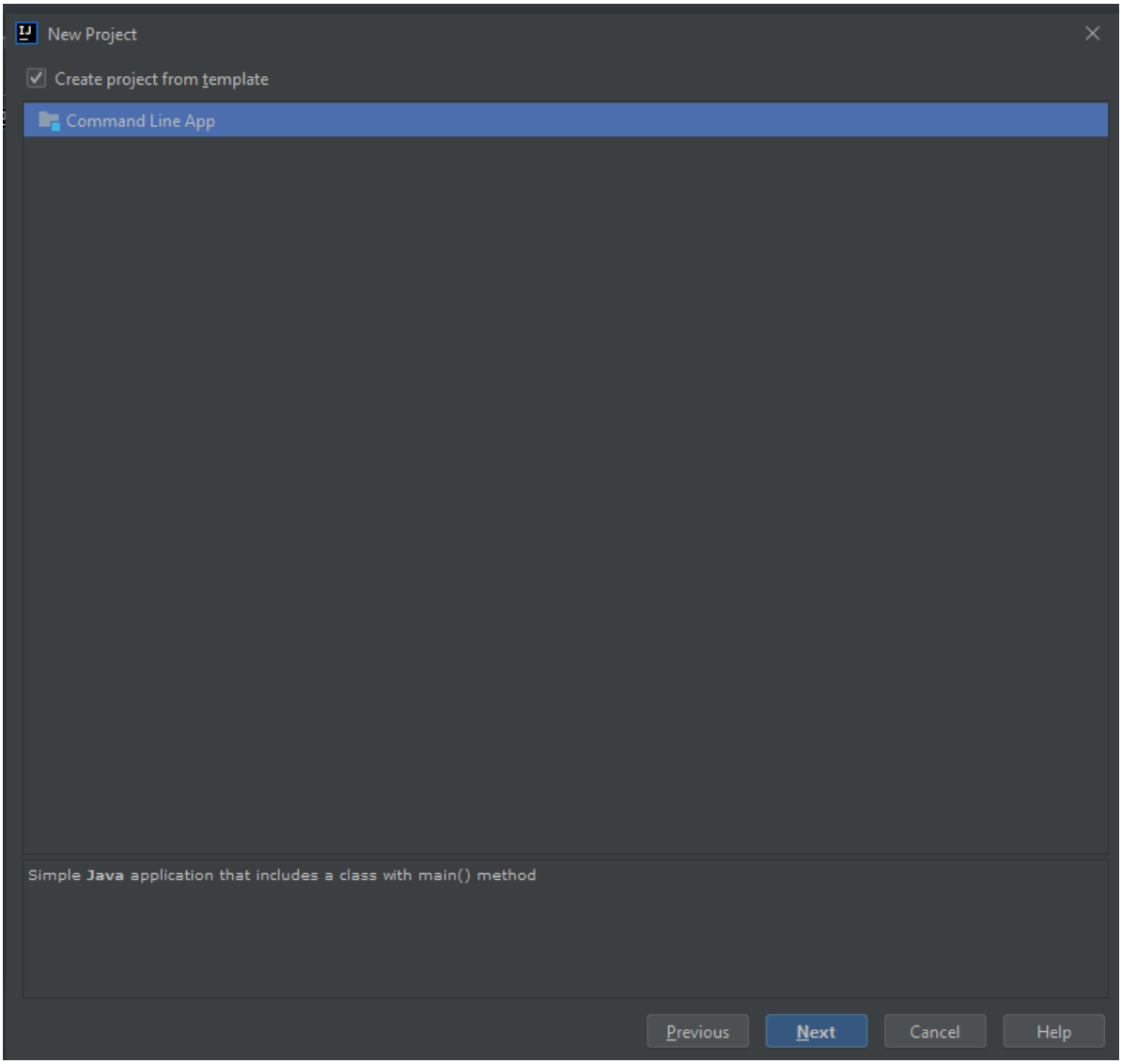
Proiect nou



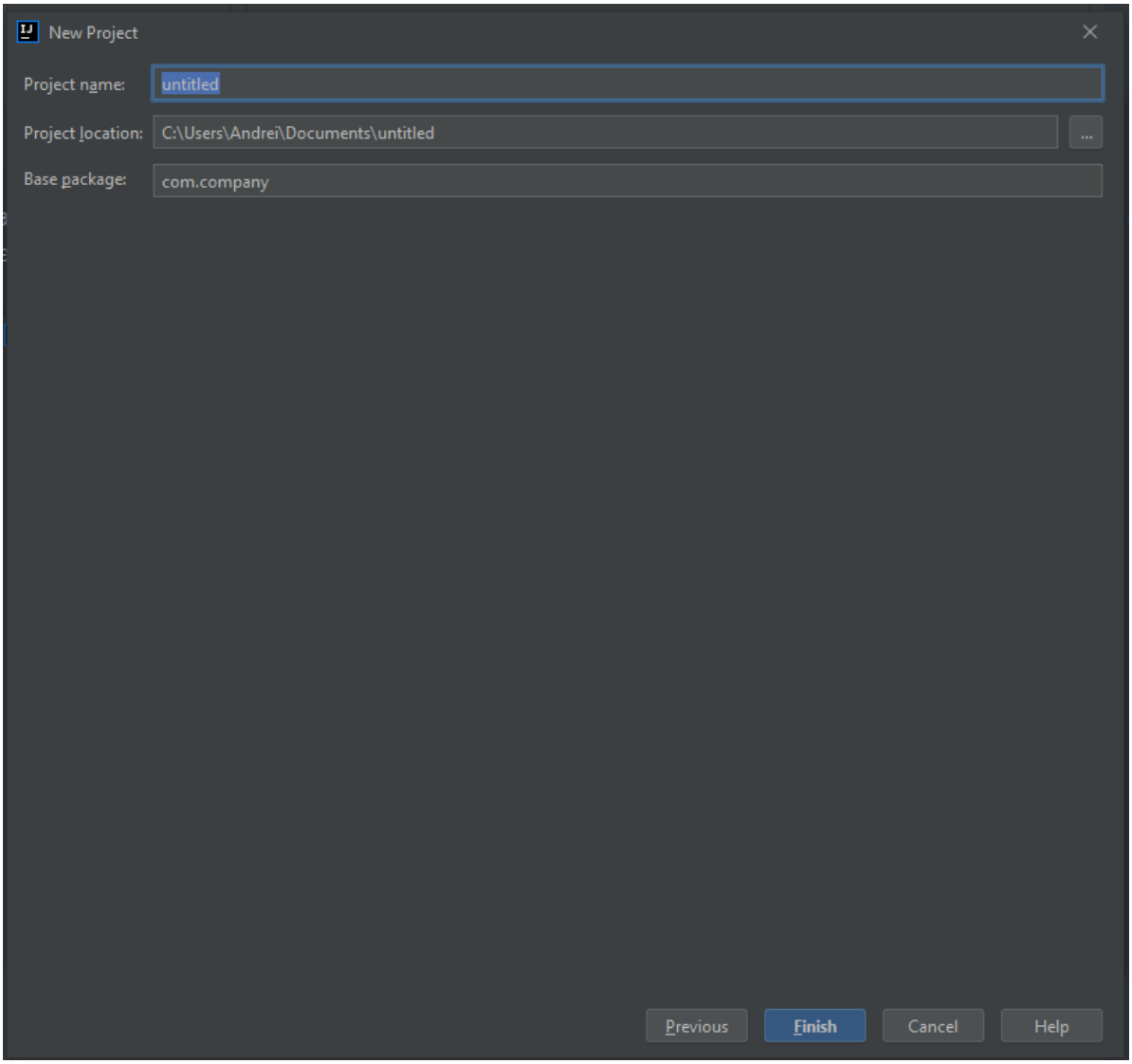
Aici selectați "New project"



Asigurați-vă că aveți Java (și nu JavaFX) selectat în partea stângă și că există un JDK existent selectat lângă „Project SDK” (este în regulă dacă numărul vostru de versiune este diferit de al meu). Acum "Next"



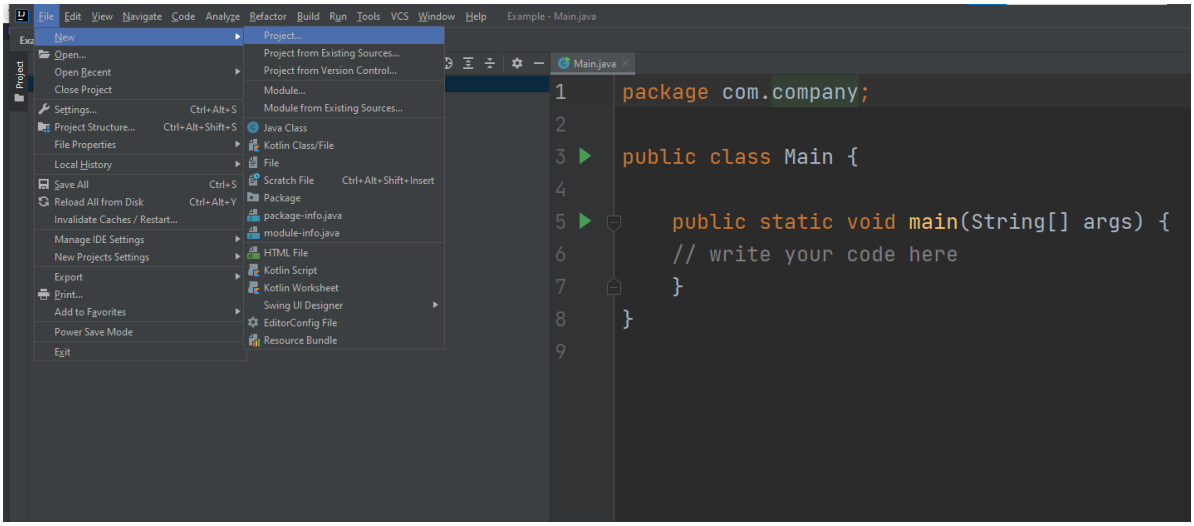
În acest pas, asigurați-vă că ați selectat "Create project from template". După ce ați făcut acest lucru, faceți clic pe "Next"



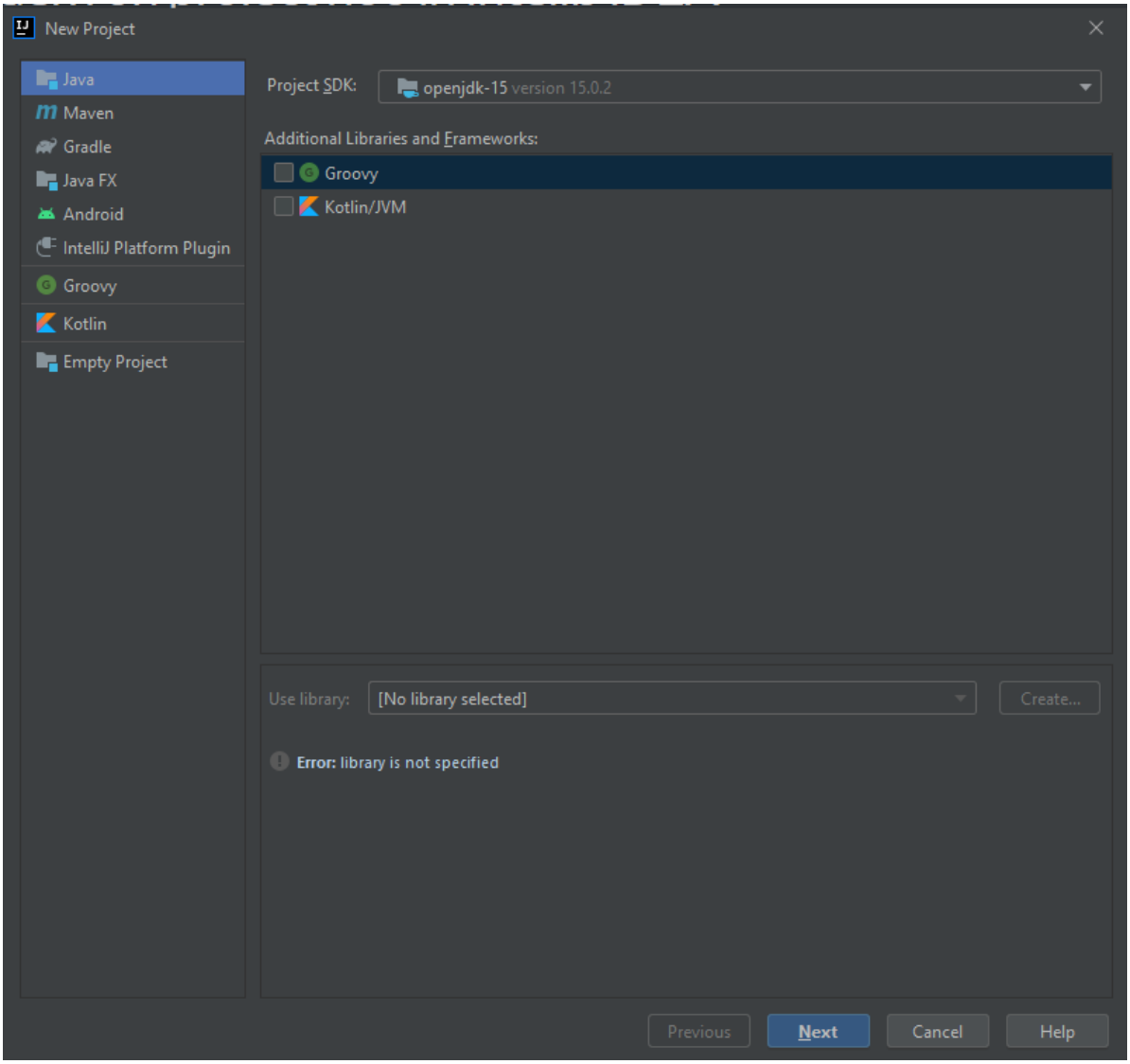
Dați aici un nume proiectului și dacă nu doriți să modificați locația în care este salvat proiectul, faceți clic pe "Finish".

Cand este deschis un proiect deja existent

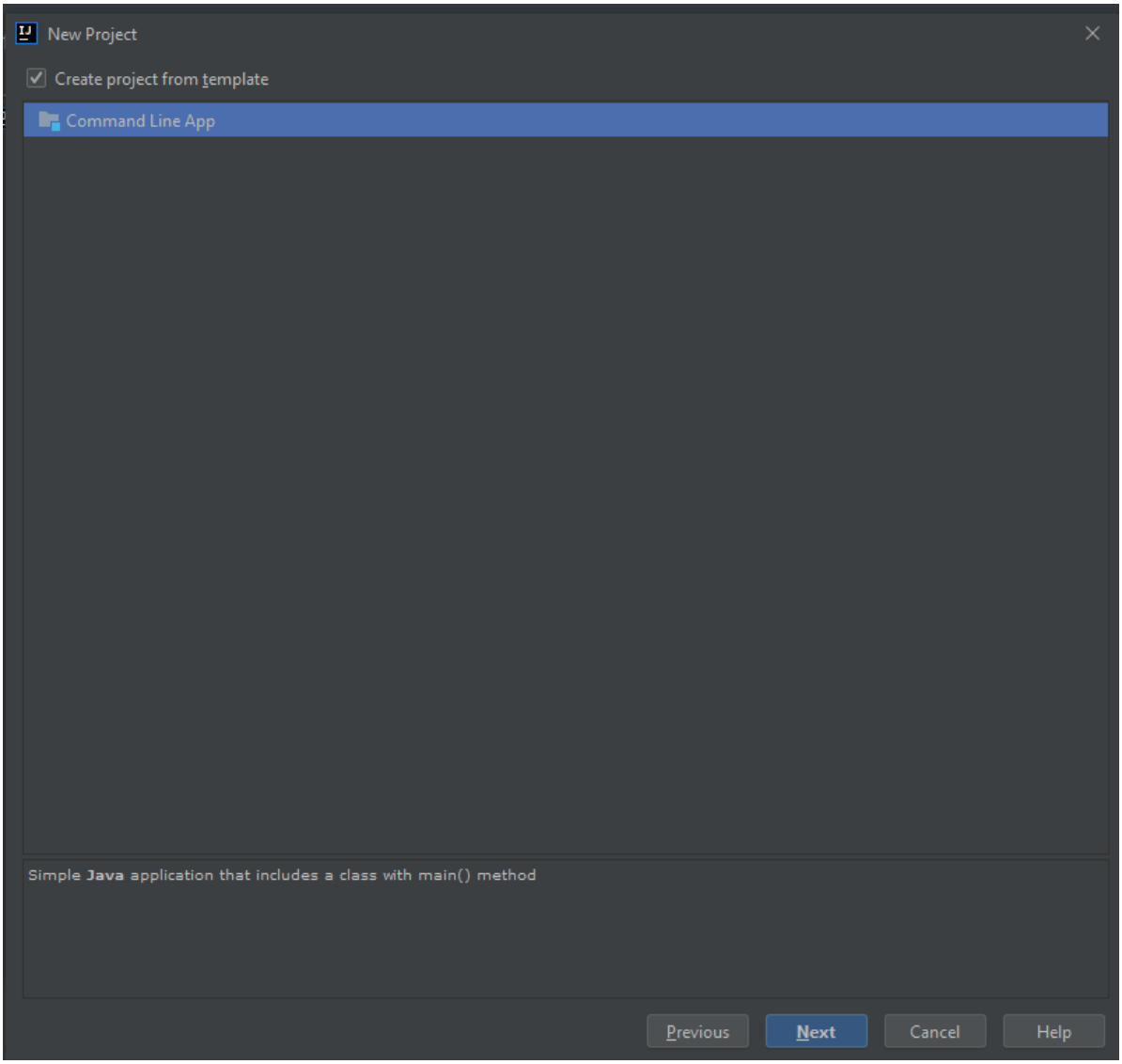
Dacă aveți deja un proiect deschis, mergeți în partea din stânga sus a ecranului și urmați pașii din screenshot. File -> New -> Project



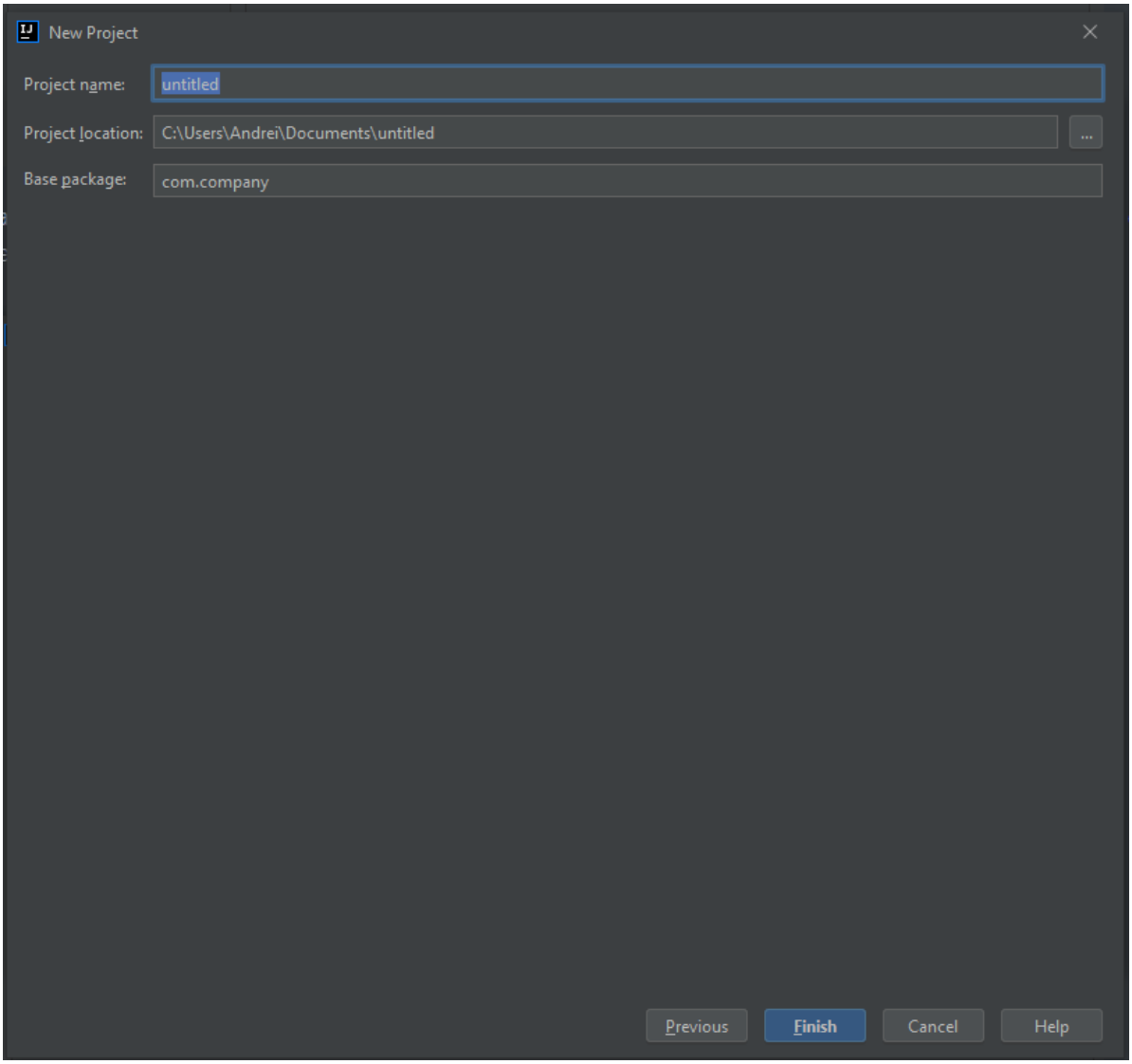
De aici, pașii vor fi aceiași cu cei pentru un nou proiect



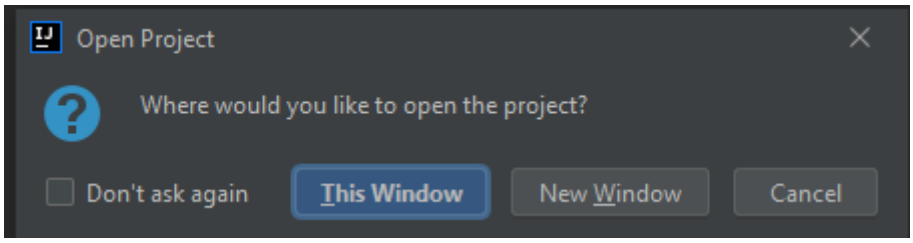
Asigurați-vă că aveți Java (și nu JavaFX) selectat în partea stângă și că există un JDK existent selectat lângă „Project SDK” (este în regulă dacă numărul vostru de versiune este diferit de al meu). Acum "Next"



În acest pas, asigurați-vă că ați selectat "Create project from template". După ce ați făcut acest lucru, faceți clic pe "Next"



Dați aici un nume proiectului și dacă nu doriți să modificați locația în care este salvat proiectul, faceți clic pe "Finish".



Pentru această fereastră, faceți momentan click pe "This Window".

Variabile

Un program manipulează (sau prelucrează) date. O variabilă este o locație de stocare (cum ar fi o cutie) care stochează o bucată de date pentru procesare. Se numește variabilă, deoarece puteți modifica valoarea stocată în interior. Mai precis, o variabilă este o locație de stocare numită, care stochează o valoare a unui anumit **tip de date**. Cu alte cuvinte, o variabilă are un **nume**, **un tip** și **stochează** o valoare de acel tip special.

Reguli de numire a variabilelor

Atunci când creați variabile, vă rog să utilizați un nume adecvat care sugereaza scopul pe care îl servește în cod. De exemplu:

```
int a = 24;
```

Încercați să aflați care este scopul variabilei de mai sus. Asa cum e acum, ne putem da seama că este o variabilă `int` și deține valoarea 24, dar nu avem nicio idee la scop servește.

```
int age = 24;
```

Încearcă acum. Știți că este o variabilă `int` care deține valoarea 24 și va fi utilizată pentru a efectua unele operațiuni care se referă la vârsta utilizatorului nostru.

Camel Case, Pascal Case si Snake Case

Fiecare limbă urmează o anumită convenție atunci când își numește variabilele. Acum știm că numele variabilei trebuie să fie ceva clar, totuși ar trebui să respecte și convențiile de numire. În Java, convenția de numire pentru variabile este camelCase.

```
int camelCaseExample = 24;

int PascalCaseExample = 24;

int snake_case_example = 24;
```

Tipuri de date

În Java există 8 tipuri de date primitive care sunt prezentate mai jos:

Tipul de date	Mărimea	Descriere
byte	1 byte	Stochează numere întregi de la -128 la 127
short	2 bytes	Stochează numere întregi de la -32.768 la 32.767
int	4 bytes	Stochează numere întregi de la -2.147.483.648 la 2.147.483.647
long	8 bytes	Stochează numere întregi de la -9.223.372.036.854.775.808 la 9.223.372.036.854.775.807
float	4 bytes	Stochează numere fracționate. Suficient pentru stocarea a 6-7 cifre zecimale
double	8 bytes	Stochează numere fracționate. Suficient pentru stocarea a 15 cifre zecimale
boolean	1 byte	Stochează valori adevărate sau false
char	2 bytes	Stochează un singur caracter / literă sau valori ASCII

Tipurile de numere primitive sunt împărțite în două grupe:

Tipurile întregi stochează numere întregi, pozitive sau negative (cum ar fi 123 sau -456), fără zecimale. Tipurile valide sunt `byte`, `short`, `int` și `long`. Ce tip ar trebui să utilizați, depinde de valoarea numerică.

```
// Declararea unei variable de tip byte
byte myByteNumber = 127;

// Declararea unei variable de tip short
short myShortNumber = 32767;

// Declararea unei variable de tip int
int myIntNumber = 2147483647;

// Declararea unei variable de tip long
long myLongNumber = 9223372036854775807;
```

i De-a lungul cursului vom folosi cel mai mult tipul de date `int` pentru comoditate, cu toate acestea, celelalte tipuri de date sunt utilizate în cazurile în care avem nevoie de ceva specific, de exemplu, economisirea memoriei pentru numere mai mici utilizând tipul de date de `byte`.

Tipurile cu virgulă reprezintă numere cu o parte fracționată, conținând una sau mai multe zecimale. Există două tipuri: `float` și `double`.

```
// Declararea unei variable de tip float
float myFloatNumber = 3.1415f;

// Declararea unei variable de tip double
double myDoubleNumber = 3.1415;
```

⚠ Când creați o variabilă `float` în Java, vă rog să notați f-ul final după număr. Acest lucru îi spune în mod explicit că ar trebui să fie un `float`, altfel limbajul îl va interpreta ca input pentru o variabilă de tip `double`.

Tipurile `char` si `boolean`

```
// Declararea unei variable de tip char
char myCharacter = 'a';

// Declararea unei variable de tip boolean
boolean myBoolean = true;
```

Vă rugăm să rețineți că la inițializarea unui `char` cu o valoare, litera trebuie să fie între ghilimele unice.

Tipul de data `String`

Vom vorbi acum despre un alt tip de date care în acest caz nu mai sunt date primitive. Tipul de date în cauză se numește `String` și conține un sir de caractere.

```
// Declararea unei variable de tip String
String myName = "Andrei";
String mySentence = "This is a sentence I have made to show you just how much
text can be stored in a string."
```

Vă rog să rețineți că la inițializarea unui `String` cu o valoare, litera trebuie să fie între ghilimele duble.

Comentarii

Vom vorbi acum despre comentarii, după cum probabil ați văzut în exemplele de mai sus, am folosit comanda `//` urmată de o explicație. Acestea se numesc comentarii. Comentariile **nu** sunt executabile și sunt ignorate de compilator (vom discuta si noi mai tarziu ce este un compilator). Dar ele oferă explicații și documentații utile cititorilor (și pentru tine, când te întorci la muncă luni). Există două tipuri de comentarii:

`//` - Single-line comments (linie unică): începe cu `//` și durează până la sfârșitul liniei curente.

`/*....*/` - Multi-line comments: începe cu `/ *` și se termină cu `* /` și poate cuprinde mai multe linii.

System.out.println()

`System.out.println ("Hello World!")`

Este folosit pentru a printa șirul "Hello World!" in terminal. Un șir este înconjurat de o pereche de ghilimele duble și conține texte. Textul va fi afisat așa cum este, fără ghilimele duble. O instrucțiune de programare se încheie cu un punct și virgulă (;).

În ceea ce privește primul program, vă rugăm să copiați și să inserați codul de mai jos.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Dacă utilizați IntelliJ IDE pentru a rula programul, puteți utiliza comanda de la tastatură `Shift` + `F10`.

Puteți utiliza `System.out.println()` (print-line) sau `System.out.print()` pentru a afisa mesaje text in consola de afișare.

- `System.out.println("Text")` (print-line) afiseaza "Text" și mută cursorul la începutul liniei următoare.
- `System.out.print("Text")` afiseaza "Text", dar plasează cursorul după șirul afisat.
- `System.out.println()` fără nimic intre paranteze afiseaza o linie nouă.

Vă încurajez acum să testați rezultatele următoarelor linii de cod și să observați diferența dintre utilizarea `System.out.println()` și `System.out.print()`

```
public class Main {
    public static void main(String[] args) {
        System.out.print("Hello"); // Aici vom folosi doar print
        System.out.print("World!");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello"); // Aici folosim println in loc de print
        System.out.print("World!");
    }
}
```

Dacă vrem să printăm valoarea unei variabile vom face așa ceva:

```
public class Main {
    public static void main(String[] args) {
        int age = 24;
        System.out.println(age);
    }
}
```

În cazul în care dorim să printăm un text și o variabilă, putem folosi concatenarea șirurilor.

```
public class Main {
    public static void main(String[] args) {
        int age = 24;
        System.out.println("Hello, I am " + age + " years old.");
    }
}
```

Vă încurajez să jucați cu concatenarea șirurilor pentru a înțelege mai bine.

Instrucțiuni repetate

În cazul în care doriți, de exemplu, să spuneți „Hello” de 100 de ori, nu va fi nevoie să copiați și să inserați aceeași comandă de mai multe ori, ci în schimb veți folosi un simplu for-loop.

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            System.out.print("Hello World!");
        }
    }
}
```

Să aruncăm o privire acum asupra acestui for-loop.

Putem observa cuvântul cheie `for`, care indică faptul că vom crea o buclă `for`, apoi deschidem parantezele și putem observa trei instrucțiuni separate fiecare de câte un `;` le vom analiza acum pe fiecare:

- `int i = 0` - Aici creăm o variabilă pe care o numim `i` și o inițializăm cu valoarea 0. Vă puteți gândi la această afirmație ca și cum ați spune „Vom începe acum să numărăm de la 0
- `i < 100` - În această parte, spunem buclei când trebuie să se oprească și, în acest caz, se va opri odată ce va ajunge la 100
- `i++` - În partea finală, facem numărarea. Această linie înseamnă, practic, că la valoarea curentă a lui i vom adăuga 1 odată ce comanda din acolade este executată. Mai există și posibilitatea de a folosi `++i` care face același lucru, în sensul că aduna valoarea actuală a lui i cu 1 dar face acest lucru înaintea execuției codului din acolade.

> În Java ++ înseamnă adunarea valorii unei variabile cu 1. Este echivalentul lui `i = i + 1`

Astfel, dacă ați rulat codul de mai sus, ar fi trebuit să vedeți fraza „Hello World” de 10 ori.

O buclă `for` nu este singura buclă pe care o aveți la dispoziție, putem folosi și o buclă de tip `while` pe care o voi arăta mai jos pentru a obtine acelasi rezultat.

```
public class Main {
    public static void main(String[] args) {
        int i = 0;
        while(i < 10) {
            System.out.println("Hello World!");
            i++;
        }
    }
}
```

Deși arată diferit de bucla anterioară, dacă acordați atenție codului, acesta folosește în mare parte același principiu.

- `int i = 0` - de data aceasta am declarat variabila noastră în afara parantezelor, dar am inițiat-o tot cu 0
- `while (i < 10)` - ca în bucla anterioară, îi spunem aici când să se oprească

- `i++` - de data aceasta i ++ se află în parantezele crețate împreună cu restul codului nostru, îl așezăm acolo astfel încât de fiecare dată când codul se repeta variabila i se incrementeaza.

Vom discuta alte tipuri de bucle și modalități de reutilizare a codului în viitor. Vă încurajez să vă jucați cu valoarea lui i și cu limitele pe care le dăm pentru a învăța.

Scanner

Pentru a putea citi datele introduse de utilizator, trebuie mai întâi să folosim clasa `Scanner`. Această clasă trebuie importată din `java.util`. Ar trebui să o facă automat când tastați Scanner, dar dacă nu, plasați cursorul deasupra cuvântului roșu Scanner și apăsați `Alt + Enter` pentru Windows sau `Command + Enter` pentru MacOS. Pentru a-l folosi, urmați codul de mai jos:

```
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Please input a number: ");
        int number = sc.nextInt();

        System.out.println("The value of the number is: " + number);
    }
}
```

Când executați codul, veți vedea că terminalul se va întrerupe și așteptați să introduceți câteva date în el și apoi apăsați enter. Observați cum veți avea întotdeauna formatul de `Scanner sc = new Scanner(System.in);` așa se inițializează obiectul în sine (vom afla mai multe despre el mai târziu) după aceea vom crea apoi o variabilă în care să stocăm valoarea.

Codul de mai sus stochează valoarea unui număr întreg, pentru a stoca alte valori va trebui să utilizați diferite metode care sunt prezentate mai jos.

Metodă	Descriere
`nextBoolean()`	Citește o valoarea `boolean` introdusă de utilizator
`nextByte()`	Citește o valoarea `byte` introdusă de utilizator
`nextDouble()`	Citește o valoarea `double` introdusă de utilizator
`nextFloat()`	Citește o valoarea `float` introdusă de utilizator
`nextInt()`	Citește o valoarea `int` introdusă de utilizator
`nextLine()`	Citește o valoarea `String` introdusă de utilizator
`nextLong()`	Citește o valoarea `long` introdusă de utilizator
`nextShort()`	Citește o valoarea `short` introdusă de utilizator

Condițional (sau decizie)

Să presupunem că lucrați la o pagină web și nu doriți ca utilizatorii cu vârsta sub 15 ani să fie autorizați să o acceseze. Cum i-ați filtra? Ne-am întreba „Dacă au peste 15 ani, îi vom lăsa să acceseze site-ul nostru” asta se poate realiza folosind comanda `if`

```
public class Main {
    public static void main(String[] args) {
        int age = 14;

        if (age < 15) {
            System.out.println("You are not allowed to enter the site!");
        }
    }
}
```

Comanda `if` va verifica orice condiție definim în paranteze și dacă este adevărată, va executa codul direct sub ea dintre acolade.

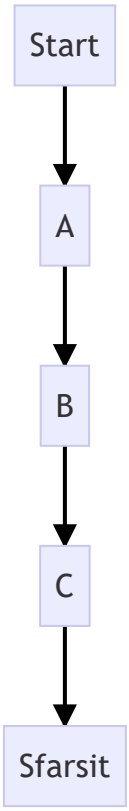
În cazul în care vă întrebați ce se întâmplă acum dacă schimb variabila de vârstă cu o valoare mai mare de 15, răspunsul este nimic. Nimic nu va fi afișat pe ecran, deoarece nu am acoperit cazul în care un utilizator are mai mult de 15 ani, astfel că va trebui să codăm această cale. Acest lucru poate fi realizat prin utilizarea cuvântului cheie `else`.

```
public class Main {
    public static void main(String[] args) {
        int age = 20;

        if (age < 15) {
            System.out.println("You are not allowed to enter the site!");
        } else {
            System.out.println("Welcome to our site!");
        }
    }
}
```

Flux de control

În general, în programare există trei tipuri de control al fluxului: secvențial, decizional/condițional și bucle.

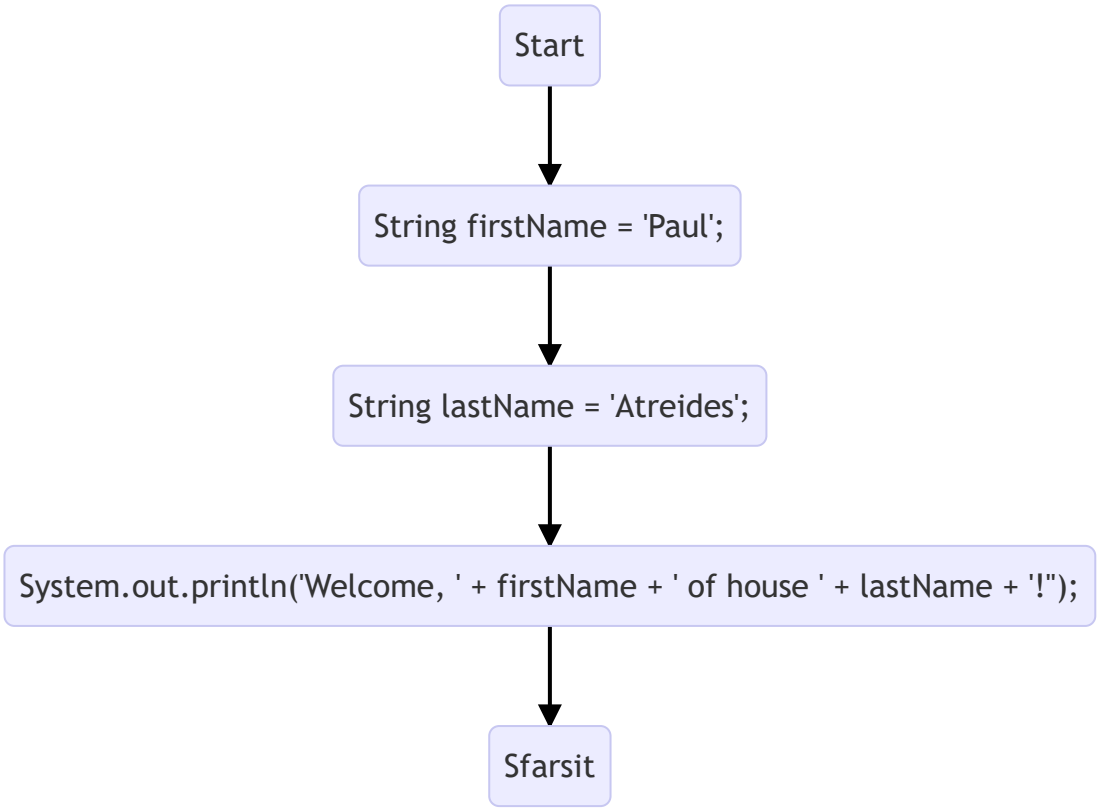
Secvențial



Un flux secvențial, așa cum arată cel de mai sus, nu este altceva decât o grămadă de instrucțiuni care sunt executate una după alta de sus în jos. Acesta este de departe cel mai comun și simplu flux pe care îl vom întâlni. Pentru a face o asociere mai bună cu diagrama de mai sus, vă rog să aruncați o privire la fragmentul de cod de mai jos.

```
public static void main(String[] args) {
    String firstName = "Paul";
    String lastName = "Atreides";
    System.out.println("Welcome, " + firstName + " of house " + lastName + "!");
}
```

În codul afișat mai sus, putem vedea că nu există alte condiții care ar putea influența modul în care este rulat codul nostru. Execuția va fi de sus în jos.



Dacă urmărim fluxul logic al codului, îl putem traduce cu ușurință într-o diagramă UML.

ℹ **Unified Modeling Language** (prescurtat **UML**) este un limbaj standard pentru descrierea de modele și specificații pentru software.

Decizional/ Condițional

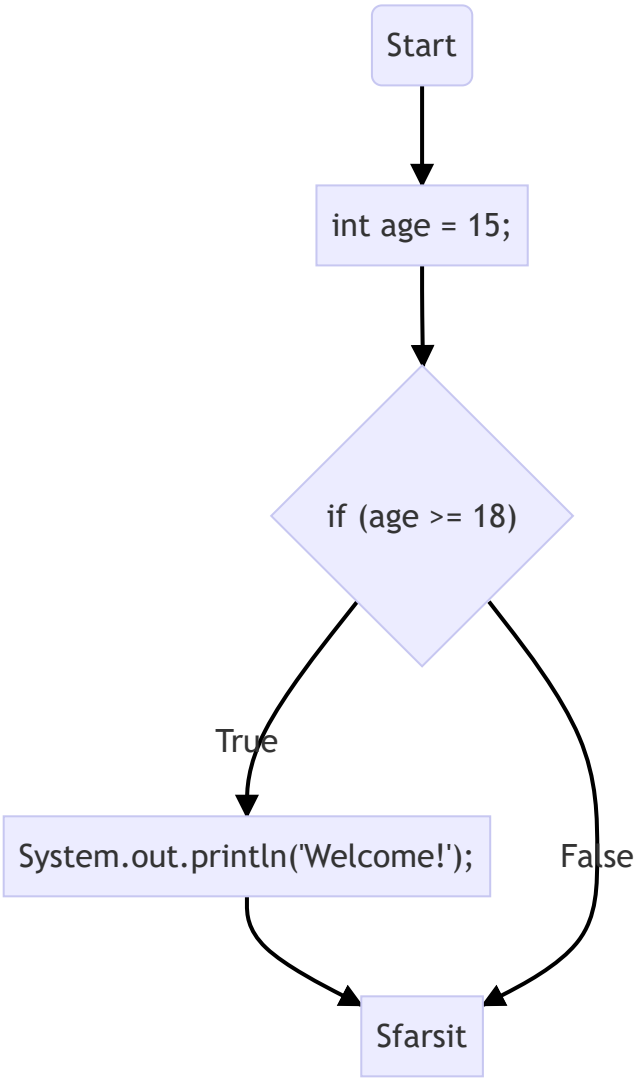
Al doilea tip pe care îl vom discuta acum sunt controalele de flux decizionale/condiționale. Acestea sunt puțin mai complexe, deoarece punem unele restricții asupra bucăților de cod care sunt executate, dar este departe de a fi complicat. Pentru a spune în termeni simpli, atunci când ne referim la fluxul condițional, ne referim la cuvintele cheie: `if`, `if - else`, `else if` și `switch`.

Condițional `if`

Cel mai simplu bloc de cod care are o condiție în el este un bloc `if`

```
public static void main(String[] args) {
    int age = 15;

    if (age >= 18) {
        System.out.println("Welcome!");
    }
}
```

Într-o diagramă UML, această logică ar arăta astfel:

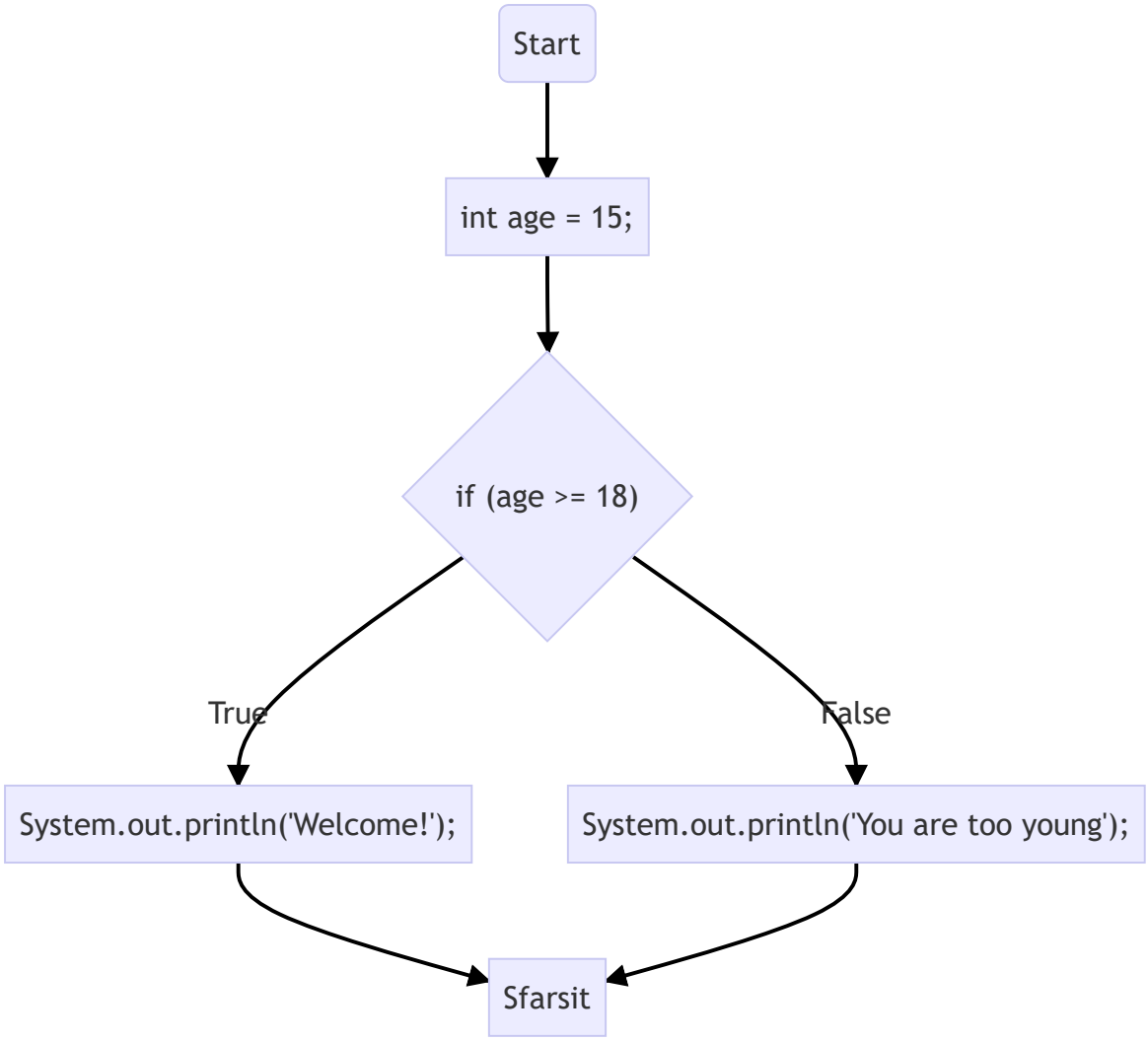


În mod similar cu modelul secvenţial, fluxul este executat de sus în jos, totuşi, când ajunge la condiţia `if`, verifică dacă este adevărat sau fals, iar în funcţie de răspuns, fie încheie programul direct, fie printează ceva pe ecran și apoi încheie programul.

Condițional `if-else`

Principala diferență de logică față de condiția anterioară este că de data aceasta, implementăm și un răspuns în cazul în care utilizatorul nostru are o vârstă mai mică de 18 ani folosind blocul `else`.

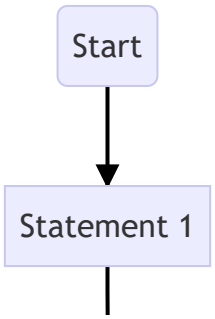
```
    public static void main(String[] args) {
        int age = 15;

        if (age >= 18) {
            System.out.println("Welcome!");
        } else {
            System.out.println("You are too young");
        }
    }
```

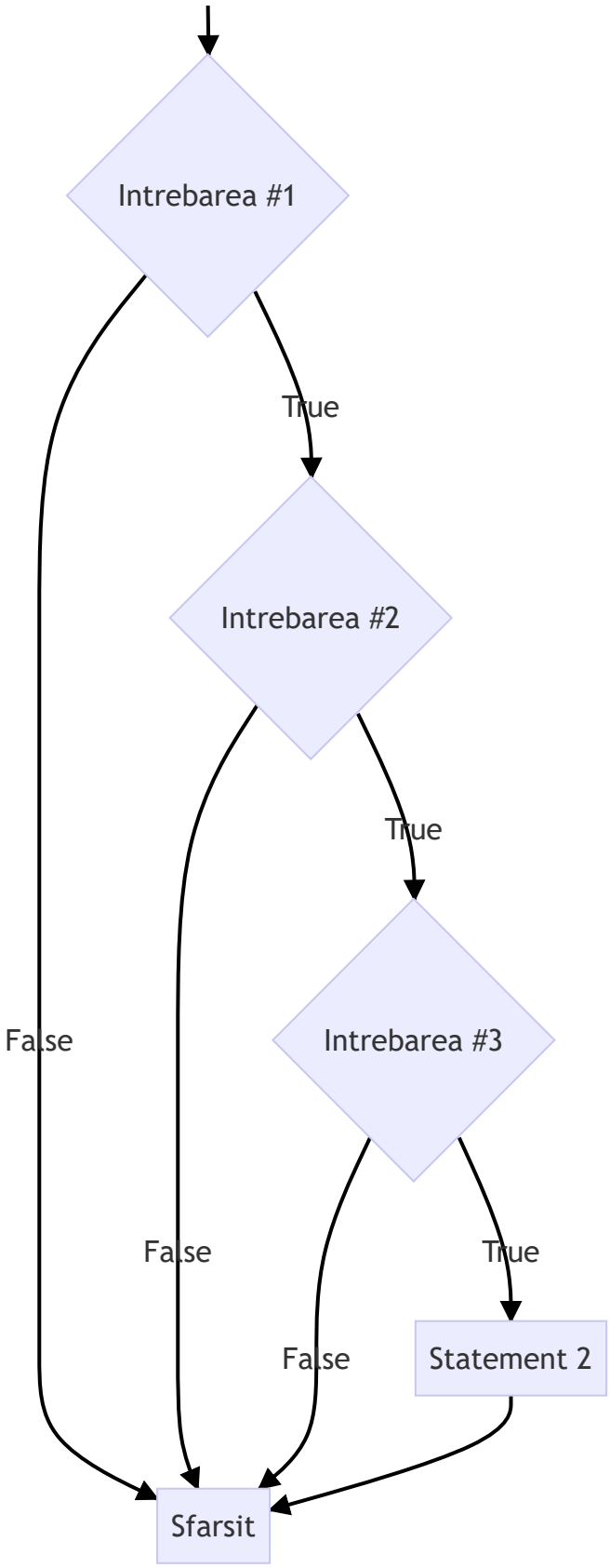


Dacă comparăm cele două diagrame, putem vedea diferența făcută de blocul `else` nou introdus. Pe scurt, blocul `else` ne oferă opțiunea de a trata orice alte cazuri care nu au fost adevărate la primele verificări.

Nested `if`

Uneori va trebui să facem declarații logice mai complexe, în care punem mai multe întrebări pentru a trata mai multe cazuri, aici intervine un `nested if`. Vă rog să aruncați o privire la diagrama logică prezentată mai jos.



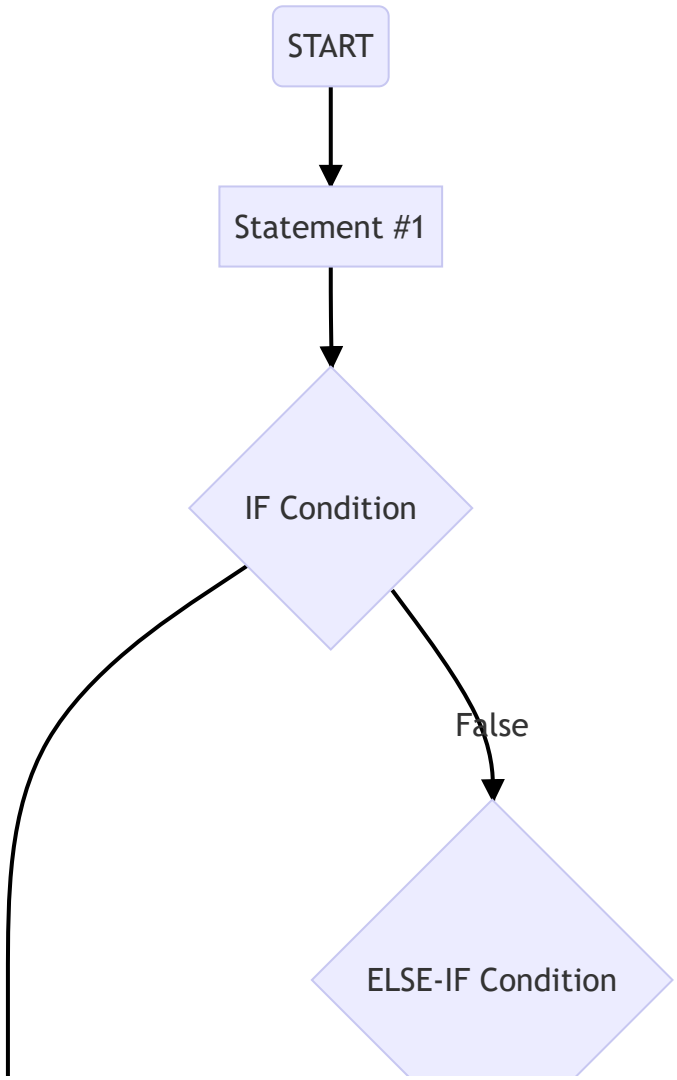


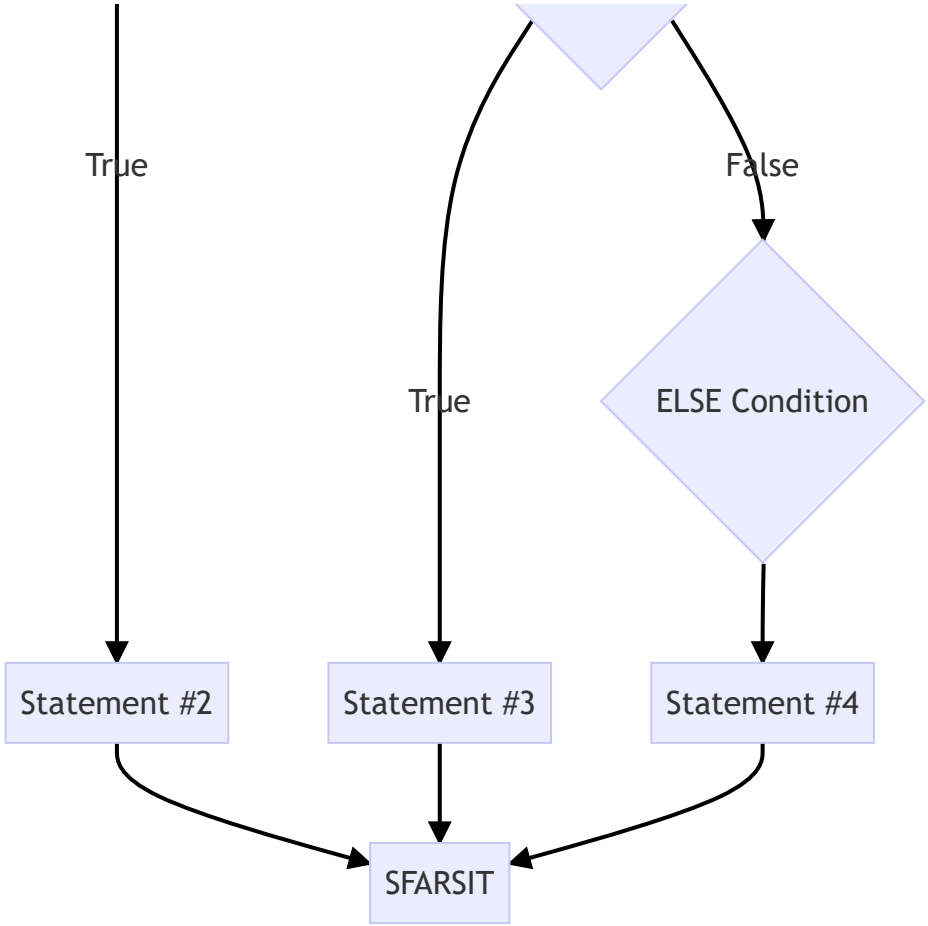
Un exemplu de cod pentru această schemă logică poate fi vizualizat mai jos. Vă rog să încercați să urmați schema logică împreună cu codul, pentru a înțelege mai bine fluxul.

```
    public static void main(String[] args) {
        // START
        int age = 15; // Statement #1

        if (age >= 18) { // Intrebarea #1 - Varsta este mai mare sau egala cu
18?
            if (age >= 20) { // Intrebarea #2 - Varsta este mai mare sau egala
cu 20?
                if (age >= 24) { // Intrebarea #3 - Varsta este mai mare sau
egala cu 24?
                    System.out.println("You are old enough for sure"); //
Statement #2
                }
            }
        }
        // SFARSIT
    }
```

Condițional `else if`

Folosim un bloc `else-if` când vrem să verificăm dacă o altă condiție este valabilă înainte de a o trimite la blocul `else`. Practic, ne permite să facem mai multe verificări pentru o condiție înainte de a o trimite la cea implicită.





La fel ca data trecută, veți avea fragmentul de cod de mai jos, așa că vă rog să încercați să îl urmați împreună cu diagrama logică.

```
    public static void main(String[] args) {
        // START
        int age = 11; // Statement #1

        if (age <= 15) { // IF Condition
            System.out.println("Statement #2");
        } else if (age <= 14) { // ELSE-IF Condition
            System.out.println("Statement #3");
        } else { // ELSE Condition
            System.out.println("Statement #4");
        }
        // SFARSIT
    }
```

Înainte de a discuta fluxul condiționat final, aș dori să vă prezint o modalitate alternativă de a scrie o condiție if care este mult mai scurtă și mai elegantă.

Expresia condițională (... ? ... : ...)

Pentru a utiliza acest formular, trebuie să aveți în vedere cum funcționează o condiție if. Deși poate părea confuz la început, logica din spatele expresiei în sine este destul de simplă. Când folosim o declarație `if`, punem o întrebare la care programul poate răspunde cu un simplu „Da” sau „Nu” (adevărat sau fals), acesta va fi primul argument pe care îl vom folosi în expresia noastră care va sta înaintea semnului „ ?".

După "?" următoarea parte a expresiei noastre este ce se întâmplă dacă întrebarea este adevărată, iar această parte se află între „?" si ":". Urmând fluxul logic al afirmației anterioare, ne putem da seama că ceea ce trebuie să punem după „:” este afirmația pentru când răspunsul la întrebarea noastră este fals.

Astfel, expresia noastră va avea următoarea formă: `intrebare ? cazul_true : cazul_false`

Mai jos este un exemplu real pe care vă sugerez să îl copiați și să îl inserați în IDE-ul vostru pentru a vă juca cu expresia și a vedea rezultatul acesteia

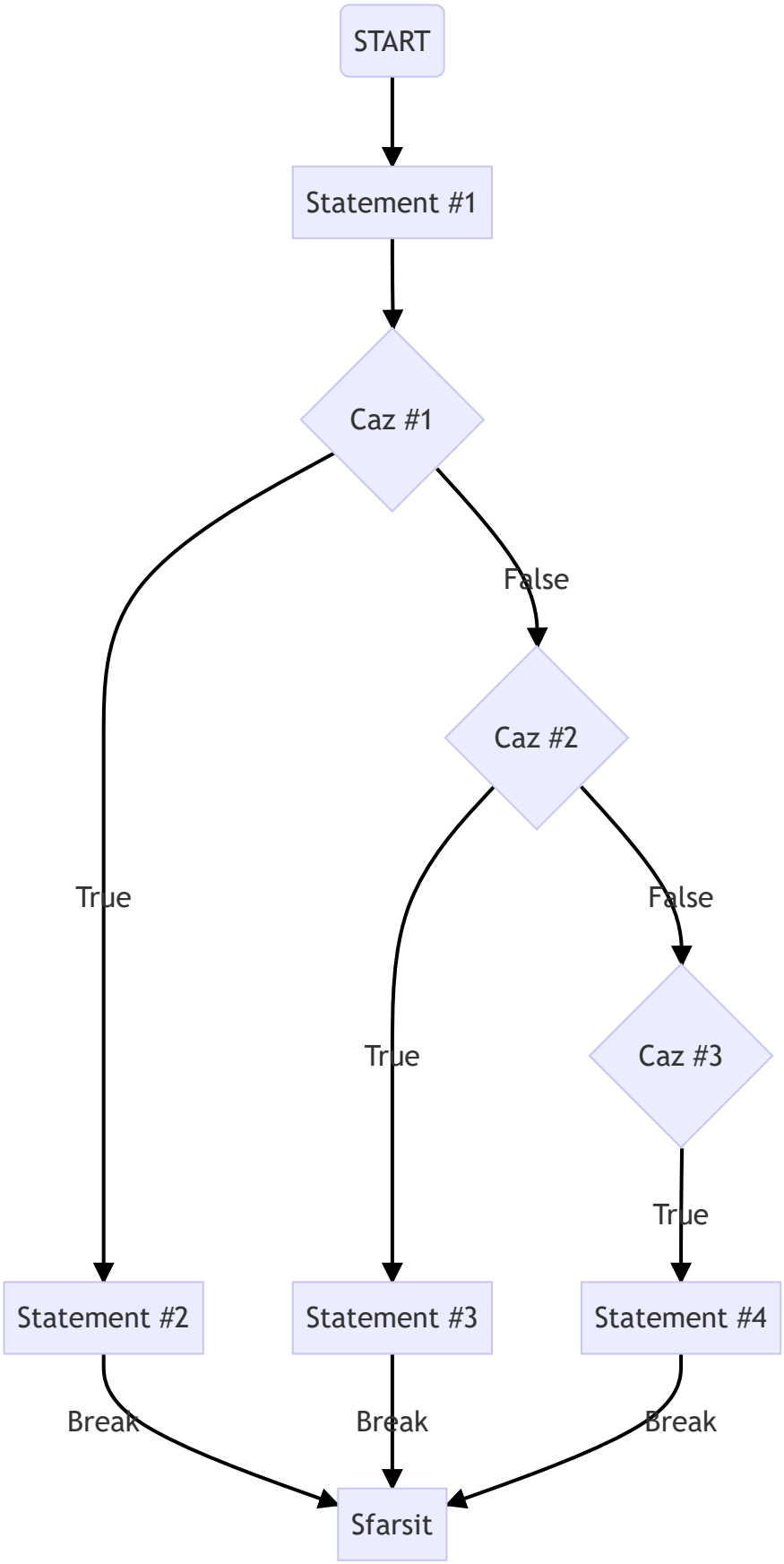
```
    public static void main(String[] args) {
        boolean isHungry = false;

        String output = isHungry ? "Go and eat something" : "Go and eat
something";
        System.out.println(output);
    }
```

Switch case

`switch-case-default` este o alternativă la `nested-if` pentru situațiile în care lucrăm cu valoarii fixe (nu se aplică pentru intervale). Puteți utiliza o variabilă `int`, `byte`, `short` sau `char` ca selector de caz, dar **NU** `long`, `float`, `double` și `boolean`. De la JDK 1.7 se acceptă și `String` ca selector de caz.

Într-un `switch-case`, este necesară o instrucțiune `break` pentru fiecare dintre cazuri. Dacă lipsește `break`, execuția va curge prin următorul caz, care este de obicei o greșeală. Cu toate acestea, am putea folosi această proprietate pentru a gestiona selectorul de valori multiple.

Ca flux logic, puteți urma schema de mai jos


START
Statement #1
Caz #1
False
Caz #2
True
False
True
Caz #3
True
Statement #2
Statement #3
Statement #4
Break
Break
Break
Sfarsit

Bucle

În programare, în general, dorim să reutilizam codul cât mai mult posibil, așa că, dacă de exemplu, ar trebui să repetăm o instrucțiune care face același lucru de 100 de ori, ar fi foarte ineficient să-l copiem și să-l lipim până ajungem la acel număr. Astfel, putem folosi ideea de buclă.

În esență, o buclă este o bucată de cod care repetă orice se află în ea până când îndeplinește o anumită condiție pentru a se opri.

Să ne uităm la schema sa logică pentru a o înțelege prin disecarea celei mai comune bucle, o buclă `for`

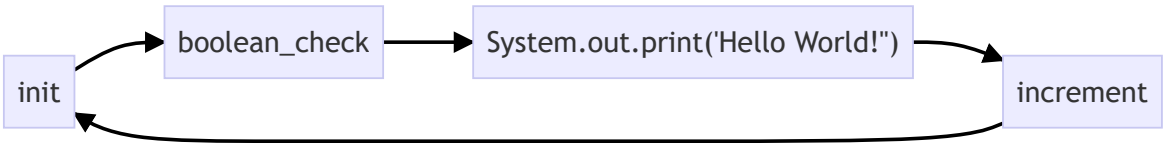
`for` loop

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            System.out.print("Hello World!");
        }
    }
}
```

Să aruncăm o privire acum asupra acestui for-loop.

Putem observa cuvântul cheie `for`, care indică faptul că vom crea o buclă `for`, apoi deschidem parantezele și putem observa trei instrucțiuni separate fiecare de câte un `;` le vom analiza acum pe fiecare:

- `int i = 0` - Aici creăm o variabilă pe care o numim `i` și o inițializăm cu valoarea 0. Vă puteți gândi la această afirmație ca și cum ați spune „Vom începe acum să numărăm de la 0. În diagramă o vom nota cu **init** de la inițializare.
- `i < 100` - În această parte, spunem buclei când trebuie să se oprească și, în acest caz, se va opri odată ce va ajunge la 100. Aici facem verificarea dacă ne putem opri din buclă sau nu. În schemă îl vom numi **boolean_check**
- `i++` - În partea finală, facem numărarea. Această linie înseamnă, practic, că la valoarea curentă a lui i vom adăuga 1 odată ce comanda din acolade este executată. În schemă îl vom numi **increment**

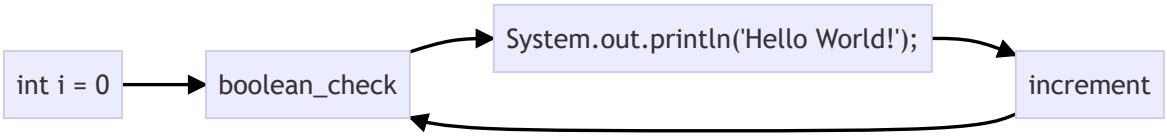


`while` loop

O buclă while va rula codul între paranteze până când îndeplinește condiția dintre paranteze declarată după cuvântul cheie `while`. Fiți atenți totuși, deoarece în cazul unei bucle `while`, trebuie să aveți grijă să declarați condiția de ieșire, deoarece altfel bucla va rula pe o perioadă nedeterminată.

```
public static void main(String[] args) {
    int i = 0;
    while (i <= 100) {
        System.out.println("Hello World!");
        i++;
    }
}
```

Codul afișat mai sus va da același rezultat ca metoda buclei `for` prezentată anterior. Deși dă același rezultat, schema logică va arăta puțin diferit.



`do while` loop

În cazul unei bucle `do while`, funcționează la fel ca o buclă `while`, cu excepția faptului că codul dintre paranteze va fi rulat cel puțin o dată. Aceasta este principala diferență dintre cele două bucle.

```
public static void main(String[] args) {
    int i = 0;
    do {
        System.out.println("Hello World!");
        i++;
    } while (i <= 100);
}
```

Observați cum codul rămâne între parantezele blocului `do`