În lecţiile anterioare aţi avut ocazia să vedeţi pentru prima dată cum este implementată logica unor programe de calculator foarte simple. Codul sursă al programelor pe care le-am scris până acum a fost executat liniar, respectiv comandă prin comandă, de sus în jos. Totuşi, aceasta nu este singura modalitate de execuţie a codului de program, astfel încât toate limbajele de programare au mecanisme diferite pentru controlul fluxului de execuţie a programului. Controlul fluxului este una dintre abordările de bază din arsenalul fiecărui programator şi, împreună cu variabilele, tipurile de date şi operatorii, formează baza fiecărui program. Prin urmare, această lecţie va fi dedicată diferitor moduri de control al fluxului în limbajele de programare populare de astăzi.

Ce este controlul fluxului?

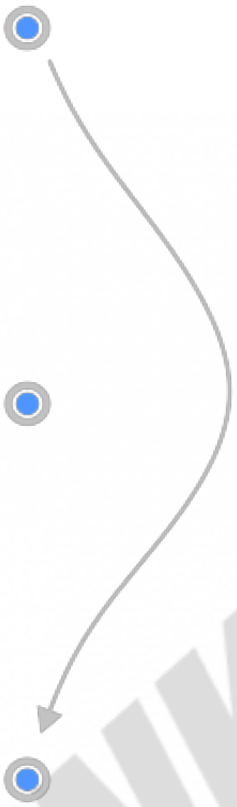
Execuţia codului de program se face de la stânga la dreapta, de sus în jos:

```
x = 1
y = 2
z = x + y
```

În exemplul prezentat, sunt definite trei linii de cod care creează 3 variabile. Acestea vor fi executate în ordinea în care sunt scrise, deci, mai întâi se va crea variabila x, apoi variabila y şi în final variabila z a cărei valoare se va obţine prin adăugarea valorilor variabilelor x şi y. În mod ilustrativ, fluxul de execuţie al codului prezentat este ca în imaginea 6.1.

Imaginea 6.1. Fluxul liniar de execuţie a codului

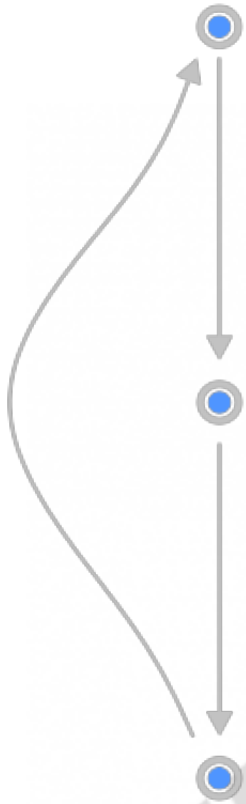
Fluxul liniar este doar o modalitate de execuţie a codului. Codul de program poate fi executat neliniar într-o varietate de moduri. Pentru început, este posibilă omiterea anumitor părţi ale codului (imaginea 6.2.).



Imaginea 6.2. Exemplu de execuţie neliniară (a controlului fluxului)

Imaginea 6.2. ilustrează o situaţie în care, după ce prima linie de cod este executată, se trece la execuţia celei de-a treia linie. A doua linie este omisă.

Pe lângă omitere, controlul fluxului se poate reflecta şi asupra repetării unor părţi ale codului (imaginea 6.3.).



Imaginea 6.3. Exemplul execuţiei neliniare (a controlului fluxului) (2)

Imaginea 6.3. ilustrează un exemplu clasic de repetare. După ce ultima, a treia linie de cod este executată, execuţia revine la început şi astfel se repetă cercul, în general, atâta timp cât este îndeplinită o condiţie.

- Pe baza a tot ceea ce este prezentat, putem rezuma că un cod de program poate fi executat în mai multe moduri diferite, deci există: execuţie liniară, selectivă şi ciclică.
- **Execuţia liniară**
  Execuţia liniară presupune execuţia codului de program în ordinea în care a fost scris. În acest fel, pot fi rezolvate doar cele mai simple probleme din timpul dezvoltării software. De

exemplu, un program, pentru a adăuga două numere, poate fi rezolvat complet folosind o structură liniară. Cu toate acestea, orice problemă mai complexă necesită o combinaţie de structură secvențială (liniară) şi de execuţie selectivă sau ciclică.

- **Execuţia selectivă**
  Abordarea selectivă permite ca anumite părţi ale codului de program să nu fie executate deloc. O astfel de execuţie selectivă permite modelarea logicii care ar trebui să conţină o decizie.
- **Execuţia ciclică**
  Aceasta este o execuţie care permite ca anumiţi paşi ai unui program de calculator să fie executaţi în mod repetat, atâta timp cât este nevoie de acest lucru. Un exemplu ideal de situaţie care ar necesita un astfel de scenariu ar fi un program de calculator al cărui scop ar fi să listeze primele n numere întregi pozitive.

Cum se realizează controlul fluxului?

Limbajele de programare de astăzi au câteva mecanisme pentru implementarea controlului fluxului. Execuţia selectivă este realizată prin ramificare, în timp ce buclele sunt folosite pentru a realiza execuţia ciclică. Aproape toate limbajele de programare moderne cunosc următoarele concepte care permit controlul fluxului de execuţie a programului:

- if,
- ..else,
- for,
- while.

Acestea sunt toate comenzile de control al fluxului cunoscute de toate limbajele de programare moderne. Aspectele lor practice vor fi

discutate mai târziu în această lecţie.

Instrucţiunea if

Instrucţiunea de ramificare de bază se numeşte instrucţiune if. Această instrucţiune există practic în toate limbajele de programare, iar scopul ei se reflectă asupra sensului ei: *dacă*.

Cuvântul `if` (dacă), de la sine, nu are prea mult sens în retorică şi este acelaşi lucru în programare. Şi anume, pe lângă acest cuvânt-cheie, trebuie să se stabilească o anumită condiţie de care va depinde rezultatul acestei instrucţiuni. Deci, if ne permite să reluăm toate acele exemple din lecţia anterioară şi, în combinaţie cu operatorii logici, operatorii de comparaţie şi tipurile de date booleene, să realizăm o execuţie selectivă.

În limbajul de programare Java, instrucţiunea `if` este formată în felul următor:

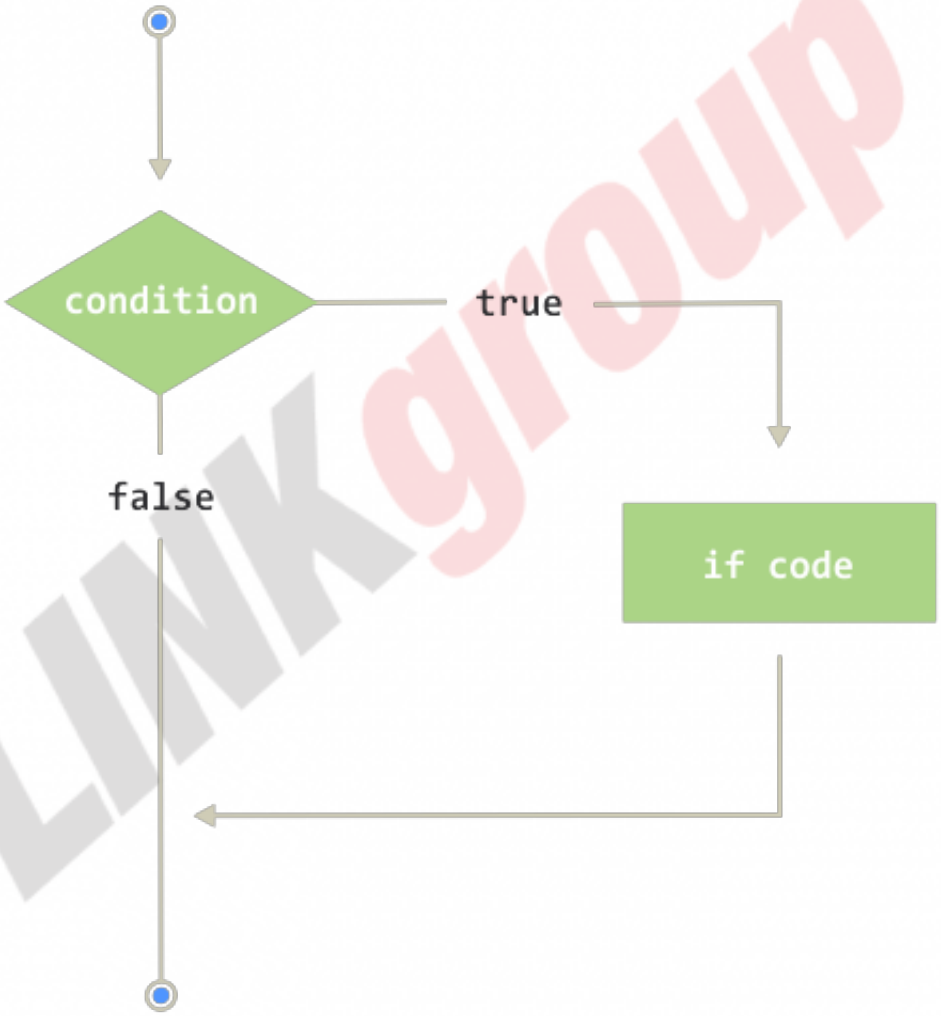
```
if(condition) {
//code to execute if condition is met
}
```

Limbajul de programare Python presupune o sintaxă ceva mai diferită:

```
if(condition):
//code to execute if condition is met
```

Diferenţa constă, în primul rând, în modul în care este format blocul de cod. Limbajul Java foloseşte parantezele acolade pentru aceasta, în timp ce limbajul de programare Python se bazează pe indentare.

Caracteristica principală a instrucţiunii if este condiţia (*condition*) de care depinde dacă codul unei astfel de comenzi va fi executat. Dacă rezultatul condiţiei are valoarea logică true, se execută codul de program al blocului if şi invers (imaginea 6.4.).



Imaginea 6.4. Instrucţiunea if

După cum se poate observa din imaginea 6.4., execuţia programului vine până la partea numită *condition* (condiţie). În acea parte se verifică condiţia definită şi, dacă este îndeplinită, se execută blocul de

cod `if`. În caz contrar, blocul de cod `if` este omis şi execuţia continuă cu prima instrucţiune următoare.

Un exemplu de utilizare a instrucţiunii if în limbajul de programare Python ar putea arăta astfel:

```
speed = 9

if (speed < 10):
 print("Too slow...")
```

Echivalentul în limbajul de programare Java arată astfel:

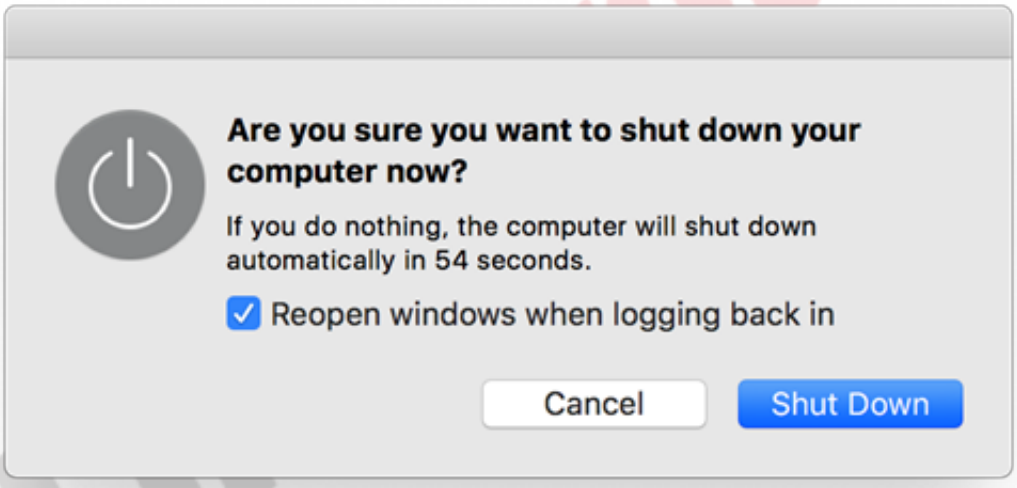
```
public class JavaHelloWorld
{

 public static void main(String[] args)
 {

     int speed = 9;

     if (speed < 10) {
       System.out.println("Too slow...");
     }

 }

}
```

La începutul exemplului este creată o variabilă cu numele `speed` şi cu valoarea `9`. După această linie urmează o instrucţiune `if`. Condiţia este definită astfel încât să se verifice dacă valoarea variabilei `speed` este

mai mică de 10. Dacă această condiţie este îndeplinită, se execută codul blocului condiţionat if, care la ieşire listează mesajul *Too slow*... Dacă din întâmplare valoarea variabilei speed a fost mai mare sau egală cu numărul 10, mesajul menţionat nu ar fi fost listat.

Exemplu de utilizare a instrucţiunii if

Un exemplu real de utilizare a instrucţiunii if este ilustrat în imaginea 6.5.



Imaginea 6.5. Instrucţiunea if

În imaginea 6.5. puteţi vedea fereastra care apare la închiderea computerului Mac. Utilizatorul are opţiunea de a alege dacă doreşte ca toate ferestrele care erau active la momentul închiderii să fie deschise la următoarea pornire a sistemului de operare. În fundal, partea logicii care este declanşată atunci când utilizatorul dă clic pe butonul Shut Down pentru închidere ar putea arăta astfel:

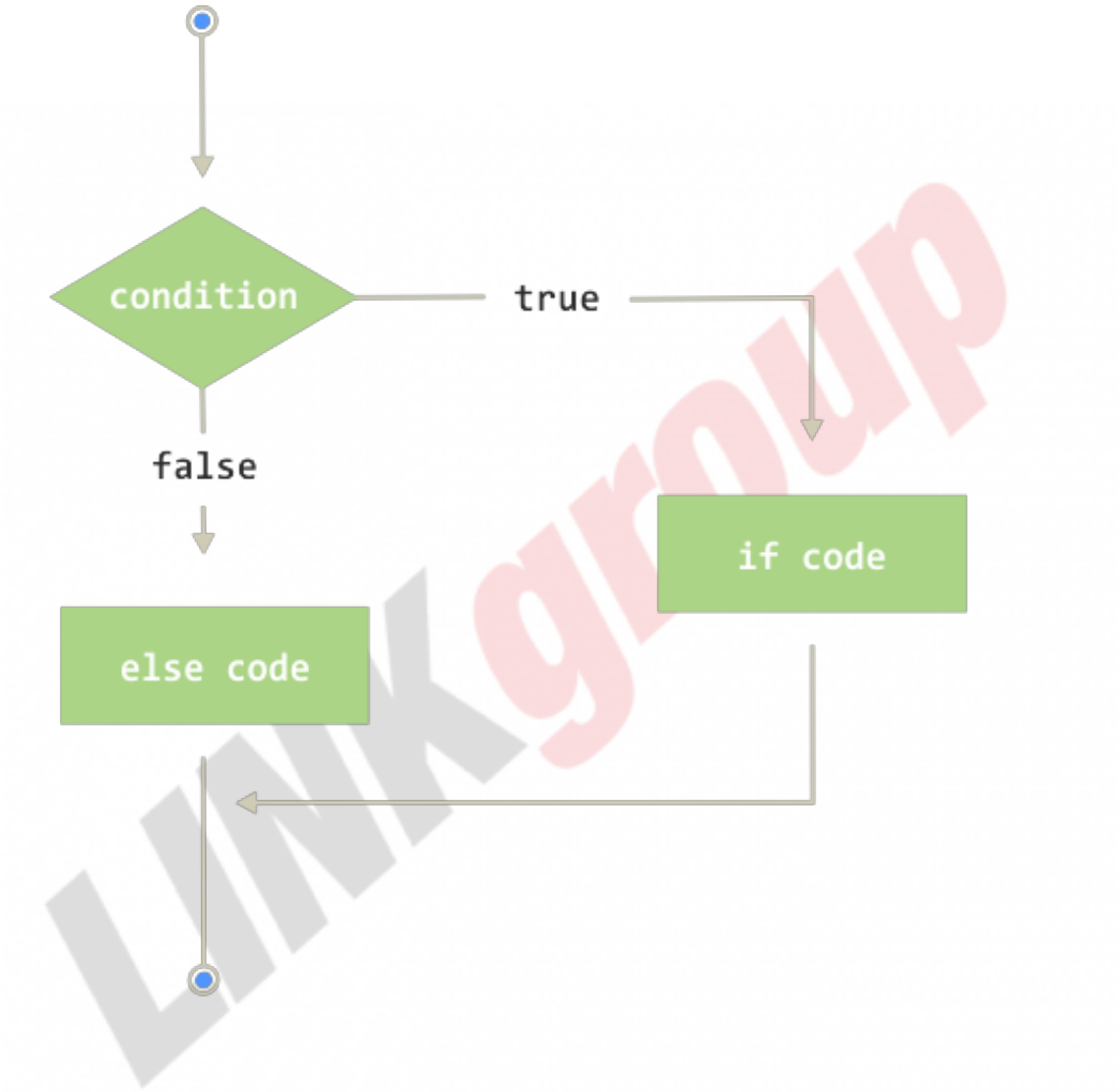
```
if(reopenCheckbox == True): saveState()
```

Codul afişat ilustrează utilizarea ramificării. Instrucţiunea if verifică dacă utilizatorul a selectat opţiunea de deschidere a ferestrelor active la următoarea pornire. Dacă a selectat-o, variabila reopenCheckbox va avea valoarea True, astfel încât starea tuturor ferestrelor active va fi salvată.

Instrucţiunea if...else

În rândurile anterioare aţi avut ocazia să vedeţi cum funcţionează instrucţiunea condiţională if, care permite execuţia sau neexecuţia anumitor coduri în funcţie de îndeplinirea unei condiţii. Când condiţia este îndeplinită, codul este executat, iar când nu este, codul este omis. Totuşi, se pune întrebarea ce se va întâmpla dacă dorim să activăm un cod specific, special pregătit chiar şi în cazul unei condiţii neîndeplinite. Într-o astfel de situaţie trebuie folosită instrucţiunea `if...else`.

Instrucţiunea `if...else` este o comandă de control al fluxului care permite definirea logicii care va fi executată dacă o condiţie este îndeplinită, dar şi dacă nu este (imaginea 6.6.).



Imaginea 6.6. Instrucţiunea If...else

Din imaginea 6.6. se poate observa că în această situaţie există două segmente de cod: *if code* şi *else code*. În funcţie de îndeplinirea condiţiei, unul dintre aceste două segmente de cod este executat. Este important să înţelegeţi că un segment de cod va trebui să fie executat. Ce segment va fi, aceasta depinde de rezultatul condiţiilor.

Iată cum ar putea arăta exemplul deja văzut, de data aceasta

actualizat cu un bloc condiţionat `else`:

```
speed = 9

if (speed < 10):
 print("Too slow...")
else:
 print("Not too slow...")
```

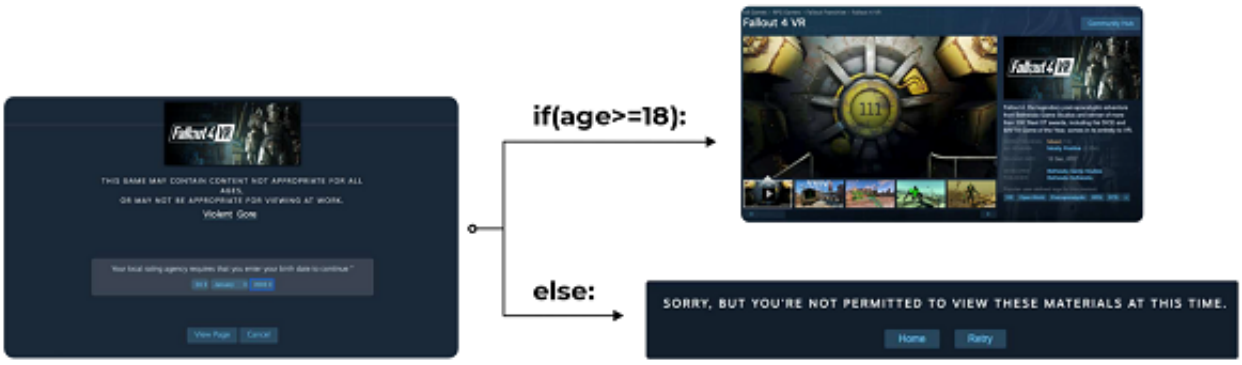
În Java acest lucru poate arăta astfel:

```
int speed = 9;

if (speed < 10) {
System.out.println("Too slow...");
} else {
System.out.println("Not too slow...");
}
```

Exemplul este acum extins cu un bloc condi?ionat `else`, astfel încât în situa?iile în care valoarea variabilei `speed` este mai mare sau egal? cu 10, va fi listat mesajul `Not too slow...`

Exemplu de utilizare a instrucţiunii if...else

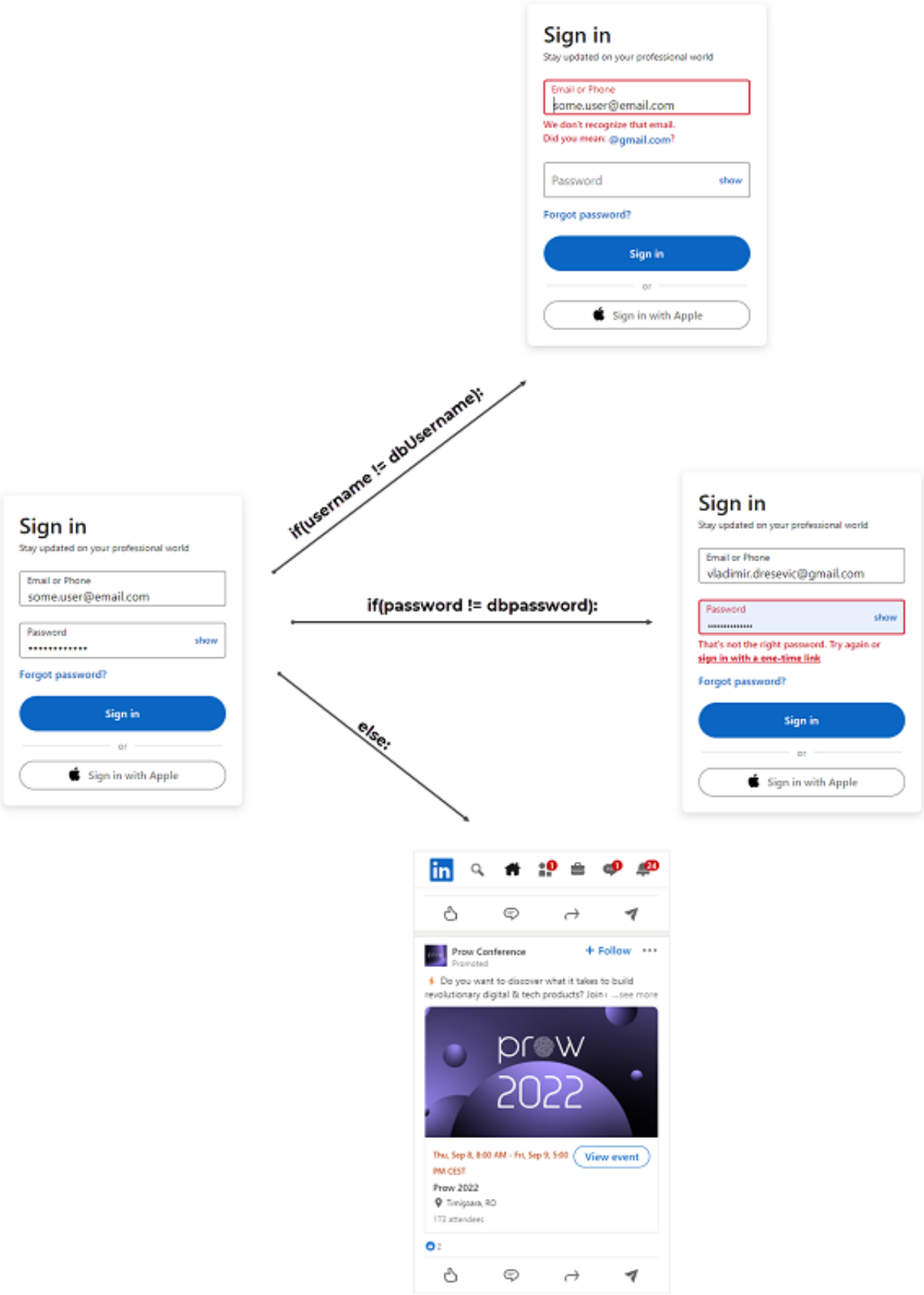
Am văzut deja în lecţia anterioară un exemplu în care este posibilă utilizarea instrucţiunii if...else. Este vorba despre verificarea vârstei jucătorului, pe baza căreia se stabileşte ce ecran va fi afişat (imaginea 6.7.).



Imaginea 6.7. Instrucţiunea If...else

În partea stângă a imaginii 6.7. puteţi vedea mesajul afişat jucătorului. Jucătorul este rugat să introducă data sa de naştere. Logica programului verifică apoi dacă utilizatorul are sub 18 ani. O astfel de verificare se formează tocmai prin utilizarea instrucţiunii if...else. În cazul în care condiţia este îndeplinită, respectiv dacă utilizatorul are 18 ani şi mai mult, jucătorului i se permite accesul. În caz contrar, blocul condiţionat else este activat, iar utilizatorul primeşte mesajul că nu poate continua jocul.

Următorul exemplu de utilizare a instrucţiunii `if...else` există pe orice site care are un formular de conectare/autentificare (imaginea 6.8.).



Imaginea 6.8. Exemplu de utilizare a comenzii if...else în timpul

autentificării

În imaginea 6.8. puteți vedea logica simplificată din spatele unui proces de verificare a datelor de acces. La început, utilizatorul introduce un nume de utilizator şi o parolă. În fundal, mai întâi este verificat numele de utilizator. Blocul condiţional if este cu siguranţă folosit pentru asta. În imaginea 6.8. puteţi vedea că se verifică dacă numele de utilizator introdus este diferit de cel care există în baza de date. Dacă da, se afişează o fereastră cu un mesaj de eroare care indică faptul că numele de utilizator introdus nu a fost găsit.

Următoarea verificare este legată de parola introdusă de utilizator. Verificarea se face după acelaşi principiu ca şi cu numele de utilizator. Se verifică dacă parola introdusă este diferită de cea introdusă în baza de date. Dacă parolele nu se potrivesc, utilizatorului i se afişează pagina corespunzătoare cu un mesaj că parola este incorectă.

În sfârşit, dacă parolele nu diferă, înseamnă că sunt identice, deci, blocul condiţionat else este activat, utilizatorul este autorizat şi i se oferă acces la aplicaţie.

Ce sunt buclele?

Spre deosebire de execuţia condiţionată (ramificare), buclele au proprietăţi oarecum diferite. Pentru a înţelege în cel mai bun mod caracteristicile lor, vom pune următoarea întrebare:

În ce mod se poate lista mesajul Hello World de 5 ori la ieşire?

Cea mai simplă soluţie pentru a obţine comportamentul descris este să specificaţi 5 comenzi succesive pentru a lista mesajul (Java):

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

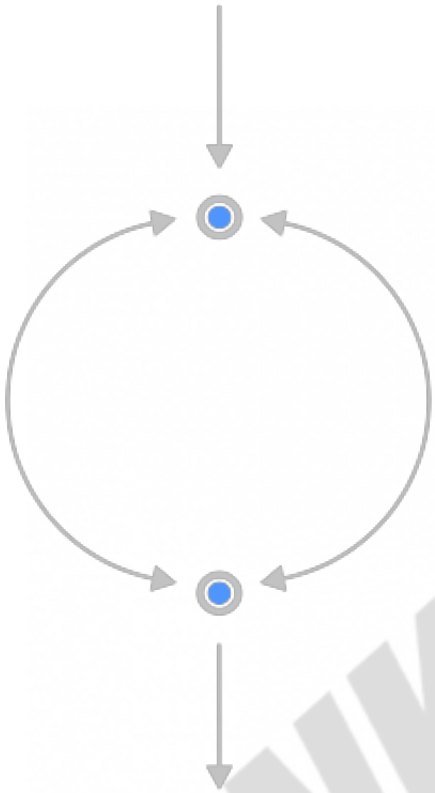
```
System.out.println("Hello World");
System.out.println("Hello World");
```

La ieşire, acest cod va produce următoarea listare:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Rezultatul dorit s-a obţinut, dar dacă ar fi necesar să listaţi de 1000 de ori mesajul `Hello World`? Listarea a 1000 de instrucţiuni identice este foarte nepractică şi plictisitoare. Ceea ce trebuie făcut pentru a rezolva problema apărută este să găsiţi o modalitate de a spune mediului de execuţie: *execută instrucţiunea pentru listarea mesajului exact de 1000 de ori*. Tocmai buclele sunt o modalitate de a formula o astfel de logică folosind codul de programare.

Logica de bază a buclelor este ilustrată în imaginea 6.9.



Imaginea 6.9. Buclă

Din imaginea 6.9. se poate concluziona că, în lumea programării, buclele permit ca o anumită parte de cod să fie executată de mai multe ori, în aşa fel încât fluxul de execuţie se reîntoarce în mod ciclic.

La baza fiecărei bucle se află un proces de repetare. Fiecare repetare se numeşte altfel **iteraţie**. Bucla începe cu prima iteraţie şi se termină după ultima iteraţie. Numărul de iteraţii ale unei bucle poate fi arbitrar, de la 0 la infinit. O buclă cu un număr infinit de iteraţii nu are sfârşit, aşadar, o altfel de buclă se numeşte **buclă infinită**.

Există mai multe bucle diferite în lumea programării. În continuare ne vom concentra pe două astfel de bucle:

- for,
- while.

Bucla for

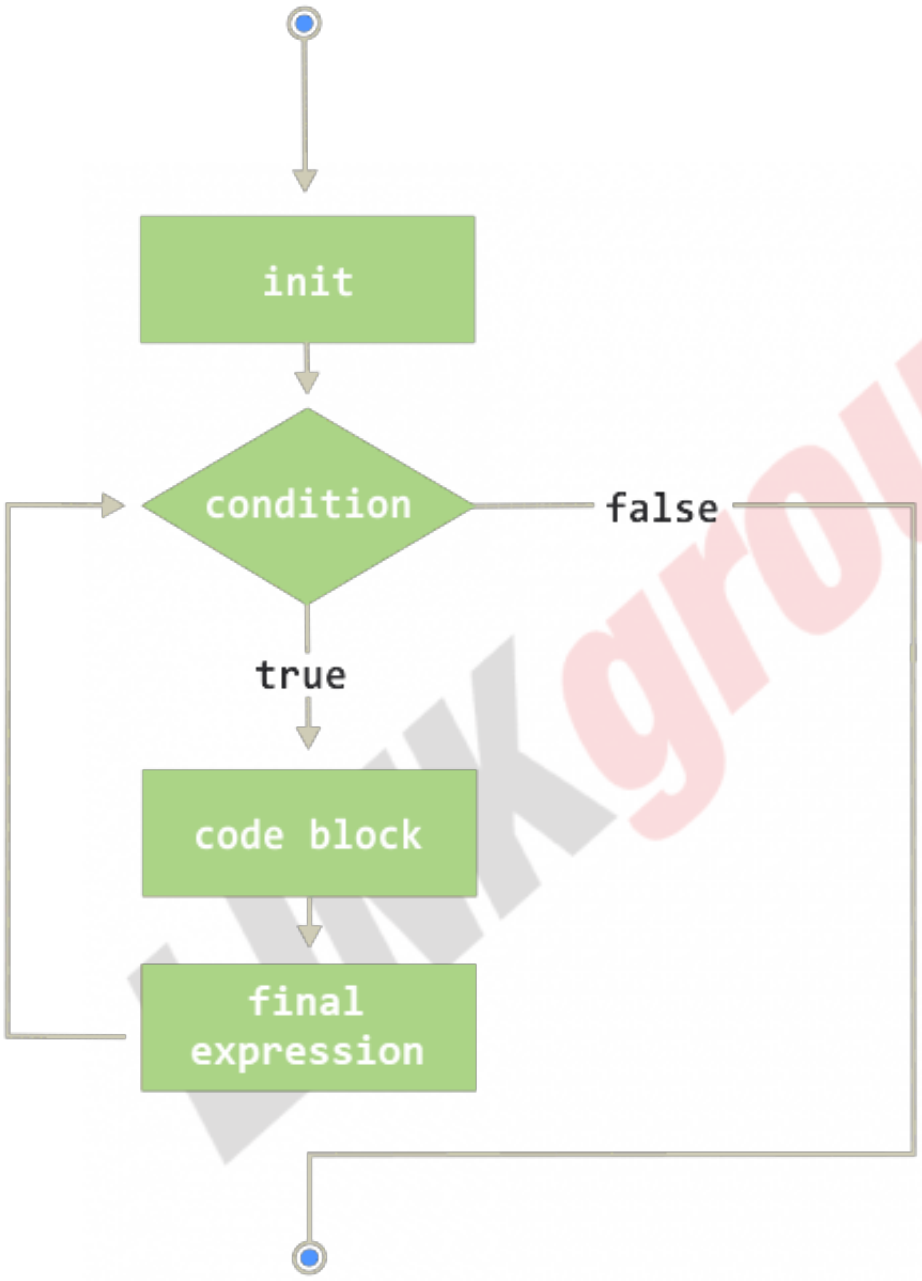
O buclă `for` permite ca un anumit bloc de cod să fie executat de un număr exact de ori. Iată cum utilizarea unei astfel de bucle poate rezolva problema din exemplul anterior, respectiv cum se poate lista mesajul `Hello World` de cinci ori la ieşire (Java):

```
for (int i = 0; i < 5; i++) {
  System.out.println("Hello World");
}
```

Rezultatul codului prezentat este identic ca în exemplul introductiv:

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Pentru o înţelegere mai bună a modului în care funcţionează bucla, analizaţi imaginea 6.10.



Imaginea 6.10. Logica buclei for

O buclă for este creată folosind cuvântul-cheie `for`. După acest cuvânt-cheie, în paranteze se specifică trei instrucţiuni (comenzi):

1. **initialization (iniţializarea)** – comanda iniţială care se

execută o singură dată la începutul execuţiei buclei; se foloseşte pentru a iniţializa unul sau mai multe contoare de bucle; în exemplul prezentat aceasta este `int i = 0`;
2. **condition (condiţia)** – instrucţiune care determină adevărul condiţiei; dacă o condiţie este îndeplinită, se trece la execuţia corpului buclei; aceasta este comanda care se execută înainte de fiecare iteraţie; în exemplul prezentat, aceasta este `i < 5`;
3. **final-expression (expresia finală)** – comandă executată după fiecare iteraţie; se foloseşte, în principal, la incrementarea sau decrementarea valorilor contorului; în exemplu aceasta este `i++`.

După parantezele care conţin cele trei instrucţiuni tocmai descrise, urmează corpul buclei, care începe cu o paranteză acoladă deschisă şi se termină cu o paranteză acoladă închisă. Un număr arbitrar de instrucţiuni poate fi găsit într-un bloc de buclă care se repetă la fiecare iteraţie.

În limbajul de programare Python, un lucru identic poate fi realizat în felul următor:

```
for x in range(0, 5):
  print("Hello World")
```

Bucla while

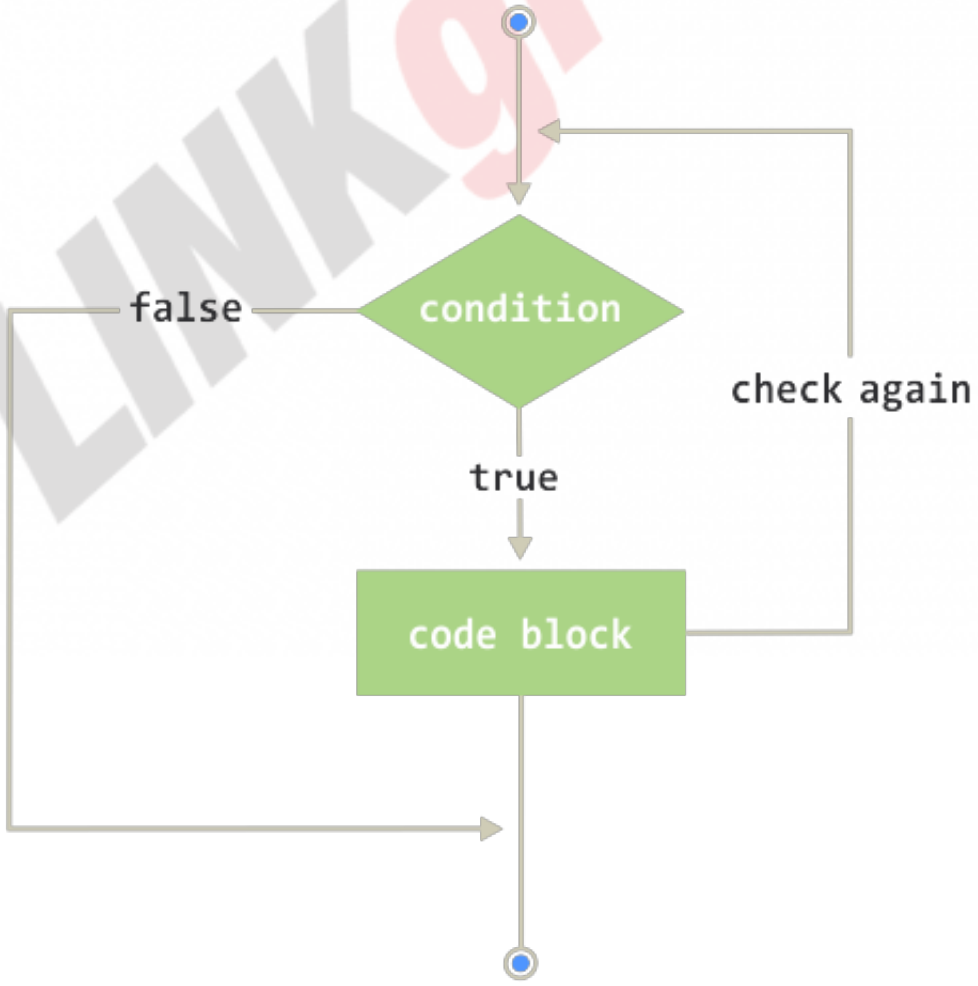
Pe lângă bucla `for` căreia i-am dedicat rândurile anterioare, această lecţie va aborda şi bucla `while`. Aceasta este o altă abordare pentru realizarea execuţiei ciclice.

Bucla `while` execută un anumit bloc de cod atâta timp cât condiţia de execuţie a buclei este îndeplinită. Acest lucru înseamnă practic că acest tip de buclă nu are un număr predefinit de iteraţii.

Un exemplu de buclă `while` în Python care poate realiza acelaşi lucru ca în exemplul anterior ar putea arăta astfel:

```
i=0
while(i < 5):
  print("Hello World")
  i = i + 1
```

Pentru a înţelege mai bine cum funcţionează bucla while, analizaţi imaginea 6.11.



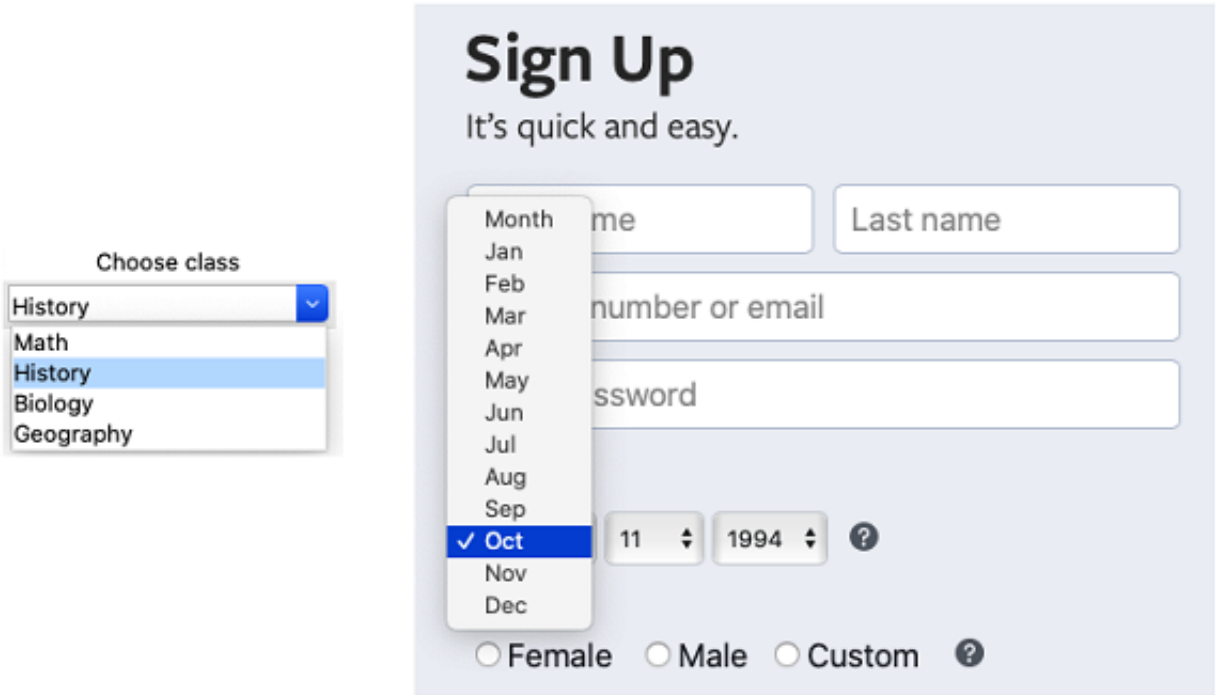
Imaginea 6.11. Bucla while

Partea centrală a buclei while este condiţia (`condition`) care este testată înainte de fiecare execuţie a codului. Dacă condiţia este îndeplinită, se trece la execuţia codului buclei. În caz contrar, ieşim din buclă.

Când se creează o buclă `while`, nu există un contor ca parte integrantă a buclei, ci de acesta trebuie avut grijă în mod independent. Prin urmare, în exemplul prezentat a fost creată o variabilă numită `i`. Este vorba de o variabilă care este folosită ca un contor în exemplu. În corpul buclei, la fiecare iteraţie, valoarea variabilei i este mărită cu unu. În acest fel se împiedică crearea unei bucle infinite.

Exemple de utilizare a buclelor

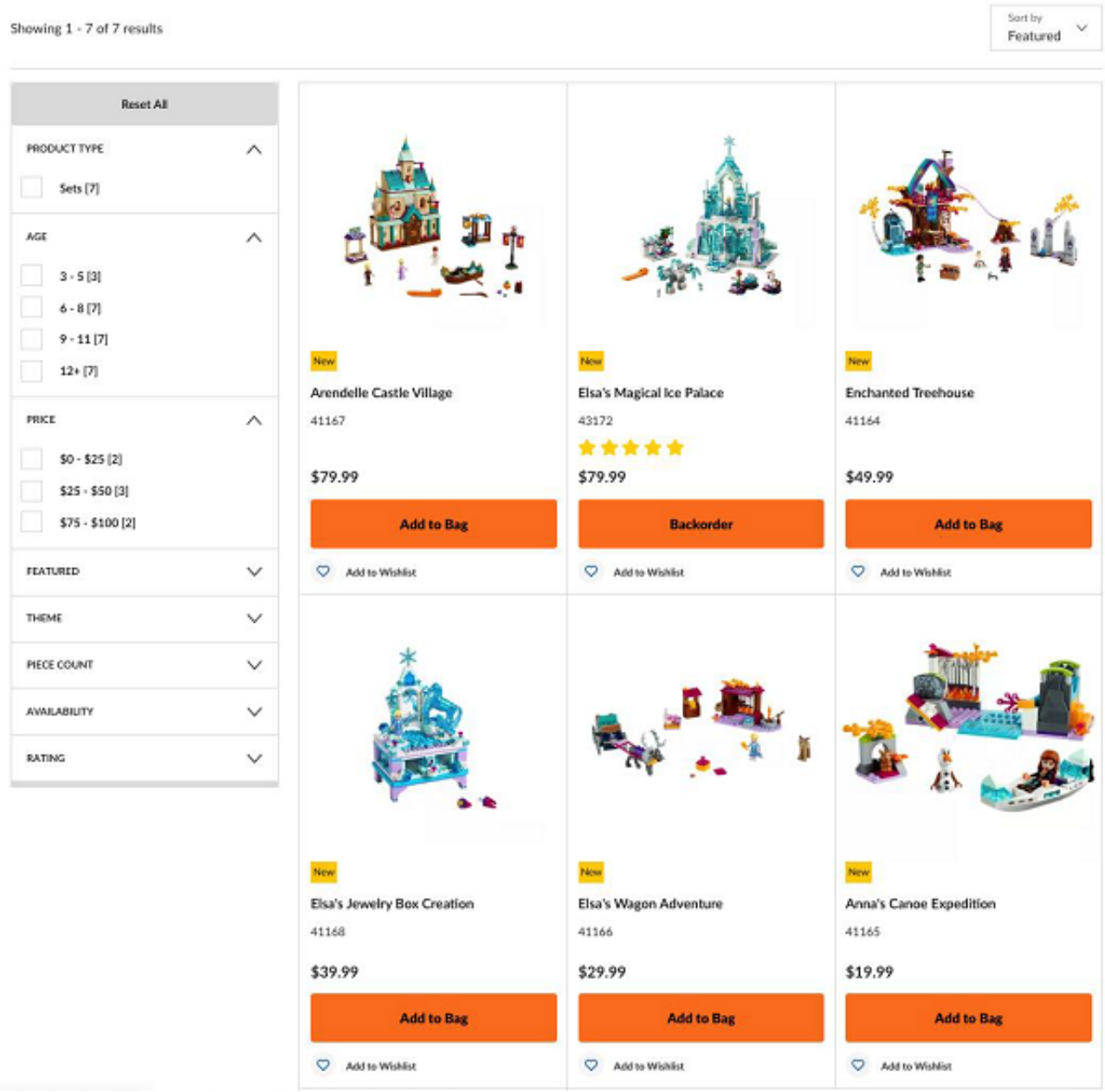
Există numeroase exemple de utilizare a buclelor. Pentru început, fiecare listă derulantă dintr-un program cu o interfaţă grafică cu utilizatorul este creată cu ajutorul buclelor (imaginea 6.12.).



Imaginea 6.12. Toate tipurile de liste se generează prin bucle

În imaginea 6.12. puteţi vedea două exemple de liste derulante. Într-una se alege subiectul dorit, iar în cealaltă, luna anului. Ambele liste sunt implementate în fundal, folosind bucle pentru a itera prin colecţia de date care, în cele din urmă, rezultă listele pe care le puteţi vedea.

Toate site-urile care au funcţionalitatea unui magazin online le permit utilizatorilor să vizualizeze produsele. Imaginea 6.13. ilustrează un astfel de site.



Imaginea 6.13. Exemplu de magazin online – în afişarea listei de produse participă cu siguranţă bucla

În imaginea 6.13. puteţi vedea o afişare clasică a produselor dintr-un magazin online. În construcţia acestei pagini şi a celor similare, a existat cu siguranţă o buclă. Lista tuturor produselor care trebuie afişate este parcursă prin buclă, apoi este creată o vizualizare a produsului în fiecare iteraţie.

Nici măcar funcţionarea jocurilor video nu poate fi imaginată fără bucle. Pentru început, grafica jocului bidimensional este alcătuită din aşa-numitele sprite-uri. Acestea sunt imagini în miniatură care se schimbă rapid în faţa ochilor jucătorului, creând astfel aspectul de animaţie. De exemplu, imaginea 6.14. ilustrează sprite-urile (părţi ale reprezentării grafice) folosite pentru a reprezenta un personaj bine-cunoscut din lumea jocurilor video.

Imaginea 6.14. Sprite-uri care formează grafica unui personaj din jocul video

Prin schimbarea rapidă a sprite-urilor, în imaginea 6.14. puteţi vedea cum se realizează aspectul mişcării. Schimbarea sprite-urilor în logica programului se realizează exact prin utilizarea buclelor. În fiecare iteraţie a buclei, este afişat un sprite. Un număr tipic de iteraţii ale unor astfel de bucle este de 20 sau mai mult într-o secundă. De fapt, termenul FPS (*frames per second* sau cadre pe secundă), care este cu siguranţă cunoscut fiecărui jucător, este legat direct de bucle.

Exerciţii

```
public class JavaProgram
{

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

```
 //todo

 }

}
```

Exerciţiul 1

În program se introduc două date întregi: 523134 şi 325423. Trebuie să se verifice restul împărţirii lui x şi y. Dacă nu există rest, atunci trebuie să afişaţi un mesaj că restul nu există, în caz contrar: trebuie să verificaţi dacă restul este mai mare sau mai mic de o mie şi să afişaţi mesajul corespunzător.

Exerciţiul 2

În program există trei variabile. O variabilă numită op reprezintă o operaţie aritmetică (+, -, *, /), iar celelalte două sunt folosite pentru a reprezenta numere. Trebuie să creaţi o funcţionalitate care, în funcţie de operaţia aritmetică reprezentată de variabila op, efectuează operaţia corespunzătoare asupra celor două variabile numerice.

Exerciţiul 3

Creaţi un program care adună primele 20 de numere întregi pozitive.

Notă

Dacă întâmpinaţi dificultăţi în rezolvarea exerciţiilor sau doriţi să

verificaţi codul pe care l-aţi scris, de la următorul link puteţi descărca soluţiile exerciţiilor: