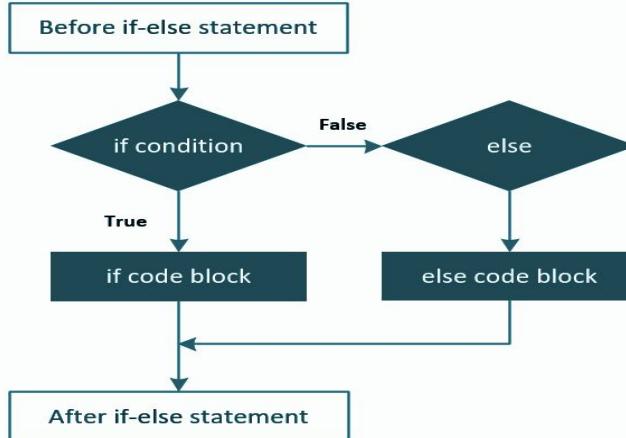


Conditionals & Control Flow: Loops in Java

Instrucțiunea if
Instrucțiunea if are două forme:

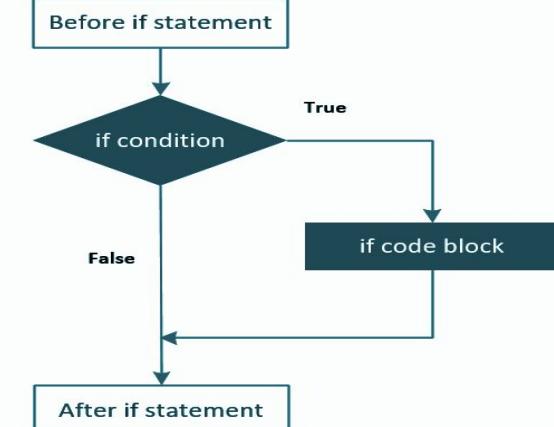
Varianta 1

```
if ( Expresie ) Instrucțiunel;  
else
```



Varianta 2

```
if ( Expresie )
```



Instrucția **if** se execută în felul următor:

- se evaluează Expresia
- dacă valoarea ei este nenulă
 - se execută Instrucție1;
 - se continuă cu instrucția care urmează după if
- dacă valoare expresiei este nulă
 - dacă există clauza else
 - se execută Instrucție2;
 - se continuă cu instrucția care urmează după if
 - dacă nu există clauza else, se continuă cu instrucția care urmează după if

Ternary Construct

You can use a ternary operator, `? :`, to define a ternary construct.

```
result = condition ? true-value : false-value;
```

A ternary construct can be compared to a compact `if-else` construct, used to assign a value to a variable depending on a boolean expression.

```
int bill = 2000;
int discount = (bill > 2000) ? 15 : 10;
System.out.println(discount); // 10
```

Correct Usage of Ternary Construct

```
int bill = 2000;
int discount = bill > 2000 ? 15 : 10;

int bill = 2000;
int discount;
discount = (bill > 2000) ? 15 : 10;

int bill = 2000;
int discount = (bill > 2000) ? bill-150 : bill - 100;

System.out.println(discount);
```

Correct Usage of Ternary Construct

A method that returns a value can also be used to initialize a variable in a ternary construct:

```
void ternaryConstruct() {  
    int bill = 2000;  
    int discount = (bill > 2000) ? getSpecDisc() : getRegDisc();  
    System.out.println(discount);  
}  
  
int getRegDisc() {  
    return 11;  
}  
  
int getSpecDisc() {  
    return 15;  
}
```

Incorrect Usage of Ternary Construct



If the expression used to evaluate a ternary operator doesn't return a `boolean` or a `Boolean` value, the code won't compile.

```
int bill = 2000;
int qty = 10;
int discount = ++qty ? 10: 20;  ✗

int discount = (bill > 2000) ? 15;  ✗
(5000 > 2000) ? 15 : 10;  ✗

int bill = 2000;
int discount = (bill > 2000) ? {bill-150} : {bill - 100};  ✗
```

Incorrect Usage of Ternary Construct

A method that doesn't return a value can't be used to initialize variables in a ternary construct.

```
void invalidTernaryConstruct() {  
    int bill = 2000;  
    int discount = (bill > 2000) ? 10 : getRegularDiscount();   
    System.out.println(discount);  
}  
  
void getRegularDiscount() {  
}
```



Instructiunea switch

switch (Selector)

{

case Constanta_1:

 Grup_Instructiuni_1;

break;

case Constanta_2:

 Grup_Instructiuni_2;

break;

... case Constanta_N:

 Grup_Instructiuni_N;

break;

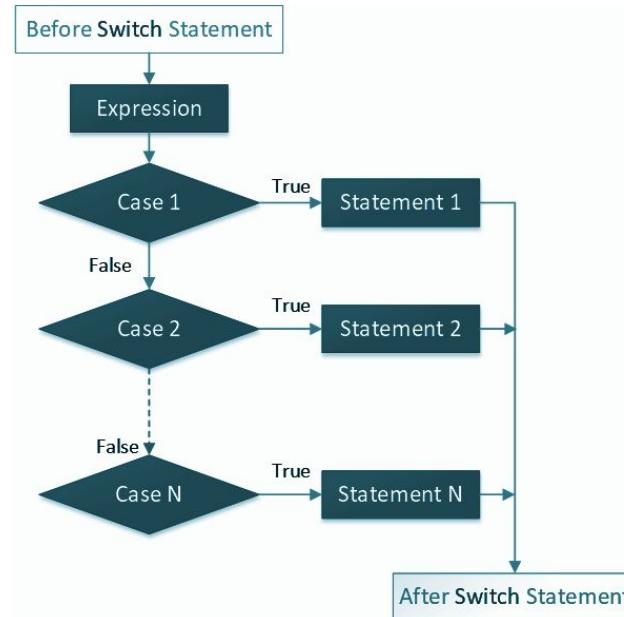
default:

 Grup_Instructiuni_default;

break;

}

Sintaxa



Use of break Statements Within a Switch

In the absence of the `break` statement, control will *fall through* the remaining code and execute the code corresponding to all the *remaining* cases that *follow* that matching case.

```
score = 50;
switch (score) {
    case 100: result = "A";
    case 50 : result = "B";
    case 10 : result = "C";
    default : result = "F";
}
```

switch statement without
break statements

```
score = 50;
switch (score) {
    case 100: result = "A";
                break;
    case 50 : result = "B";
                break;
    case 10 : result = "C";
                break;
    default : result = "F";
}
```

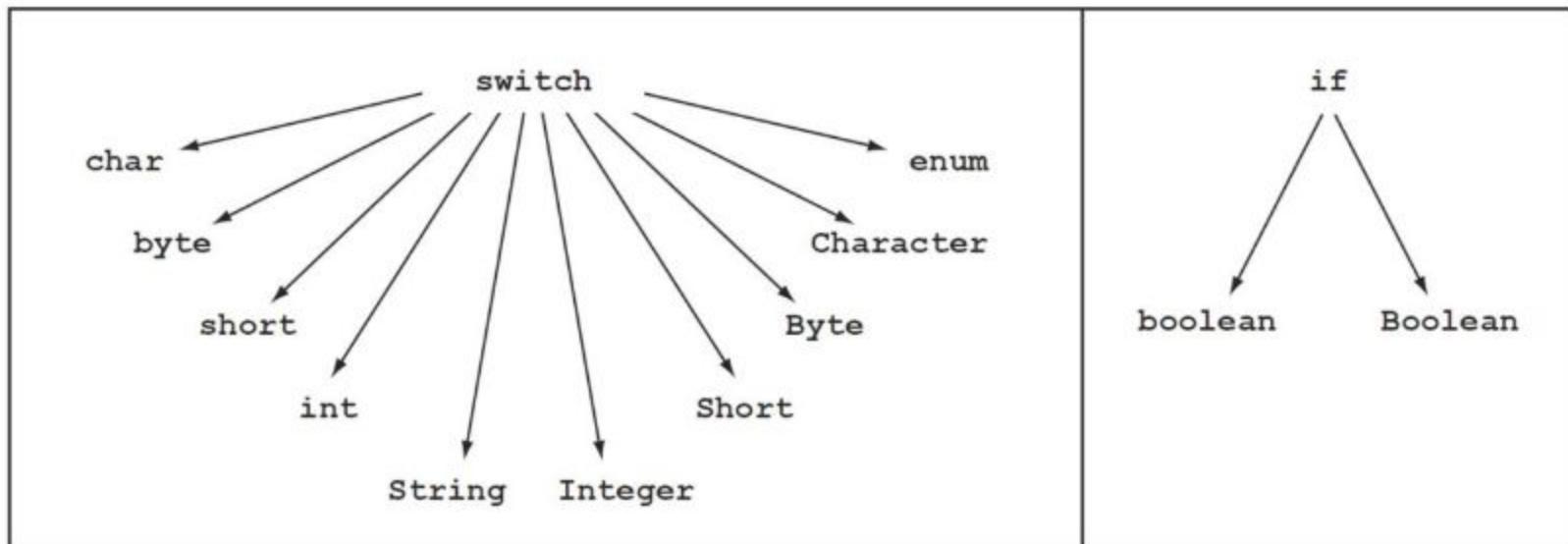
switch statement with
break statements

Instrucțiunea switch se execută în felul următor:

- Mod de execuție
- se evaluatează Selectorul
- dacă valoarea Selectorul este egală cu una dintre valorile constante din clauzele case, se execută instrucțiunile din grupul de instrucțiuni corespunzător, apoi se trece la instrucțiunea de după switch
- dacă valoarea Selectorului nu este egală cu niciuna dintre valorile constante din clauzele case, se verifică existența clausei default;
 - dacă există clauza default, se execută instrucțiunile din grupul de instrucțiuni corespunzător clauzei default, apoi se trece la instrucțiunea de după switch
 - dacă nu există clauza default, se trece la instrucțiunea de după switch

Arguments Passed to a switch Statement

You can't use the `switch` statement to compare all types of values, such as all types of objects and primitives. There are limitations on the types of arguments that a `switch` statement can accept.



What would be the output after the execution of the following code:

```
int a = 10;
```

```
int b = 20;
```

```
boolean bl = false;
```

```
System.out.print(a = b);
```

```
System.out.print(a != b);
```

```
System.out.print(bl = true);
```

Problema: Calculator aritmetic

- De la tastatură se citesc două numere reale și simbolul operației aritmetice (+,-,*./) ce trebuie efectuată. Să se afișeze la ecran rezultatul calculului efectuat.

```
import java.util.*;
public class Main
{
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        double rez=0;
        System.out.print("Indica primul numar ");
        double n1=sc.nextDouble();
        System.out.print("Indica al doilea numar ");
        double n2=sc.nextDouble();
        System.out.print("Indica operatia ");
    }
}
```

```
switch(op){
    case '+': rez=n1+n2;break;
    case '-': rez=n1-n2;break;
    case '*': rez=n1*n2;break;
    case '/': rez=n1/n2;break;
    default: System.out.println("Nu exista asa operatie");
              err=true;break;
}
```

Q2.1. What's the output of the following code?

```
int a = 10;  
  
if (a++ > 10) {  
  
    System.out.println("true");  
  
}  
  
{  
  
    System.out.println("false");  
  
}  
  
System.out.println("ABC");
```

a. true

false

ABC

b. false

ABC

c. true

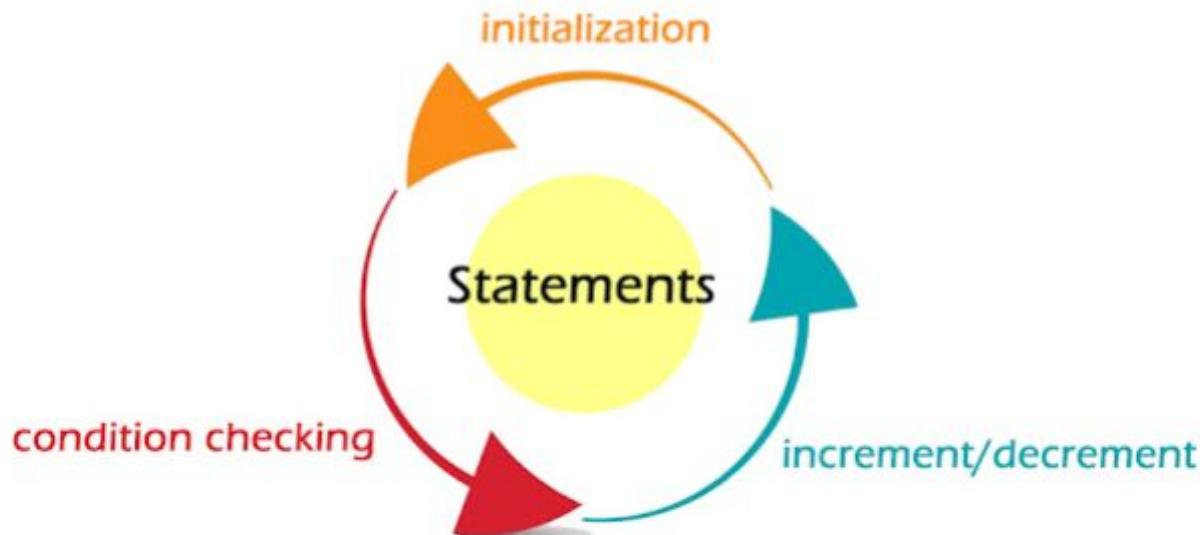
ABC

d. Compilation error

Q2.2. What's the output of the following code?

```
class EJavaGuru3 {  
  
    public static void main(String args[]) {  
  
        byte foo = 120;  
  
        switch (foo) {  
            default: System.out.println("ejavaguru"); break; a. ejava  
            case 2: System.out.println("e"); break; enum  
            case 120: System.out.println("ejava"); guru  
            case 121: System.out.println("enum"); b. ejava  
            case 127: System.out.println("guru"); break; c. ejavaguru  
        } e  
    } d. ejava  
}
```

Loops in Java



Structurile repetitive pot fi:

- **cu număr cunoscut de pași** (iterații) – se cunoaște de la început de câte ori se va execută instrucțiunea
- **cu număr necunoscut de pași** (iterații).- Instrucțiunea se execută cât timp o condiție este adevărată. La fiecare pas se va evalua condiția, și doar dacă aceasta este adevărată se va executa instrucțiunea.

Structurile repetitive cu număr necunoscut de pași pot fi:

- **cu test inițial**: mai întâi se evaluatează condiția; dacă este adevărată se execută instrucțiunea și procesul se reia.
- **cu test final**: mai întâi se execută instrucțiunea, apoi se evaluatează condiția; Dacă este adevărată, procesul se reia.

Java are următoarele structuri repetitive:

for
loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

while
loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

do-while
loop

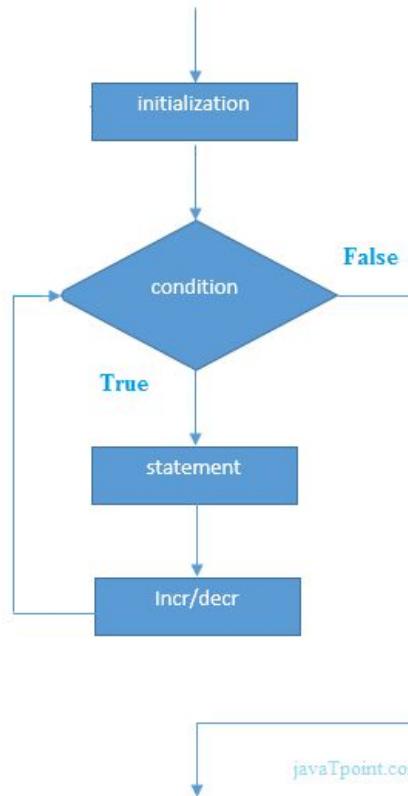
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Sintaxa

**for (Expresie_de_Initializare ; Expresie_de_Testare ; Expresie_de_Continuare)
Instructiune;**

Declaring and Initializing loop control variable
for (int i =0; i<10 ; i++) {
// Loop statements to be executed
}

Checking condition
Incrementing loop control variable



Mod de execuție

- Se evaluatează Expresie_de_Initializare
- Se evaluatează Expresie_de_Testare
- Dacă Expresie_de_Testare este nenulă:
 - Se execută Instructiune;;
 - Se evaluatează Expresie_de_Continuare.
 - Se revine la pasul 2.
- Dacă Expresie_de_Testare este nulă, se trece la instrucțiunea de după for.

Observații

- Instructiune; se execută cât timp `Expresie_de_Testare` este nenulă – condiție adevărată.
- Dacă `Expresie_de_Testare` este de început vidă, Instructiune; nu se execută deloc, iar `Expresie_de_Conditie` nu se mai evaluează.
- Instructiune; poate fi orice fel de instrucțiune, dar una singură. Dacă sunt necesare mai multe instrucțiuni, se va folosi instrucțiunea compusă.
- Este necesar ca cel puțin o variabilă care apare în `Expresie_de_Testare` să-și modifice valoarea în Instructiune; sau la evaluirea `Expresiei_de_Conditie`. Altfel se obține o **bucă infinată**.
- Cele trei expresii, `de_Initializare`, `_de_Testare` și `_de_Conditie` sunt separate prin caracterul **;** – **obiigatoriu!**
- Oricare dintre cele trei expresii, `de_Initializare`, `_de_Testare` și `_de_Conditie`, eventual toate, poate să lipsească. În acest caz avem **expresii vide**. Dacă `Expresie_de_Testare` este vidă, rezultatul său este nul!
- `Expresie_de_Initializare` se execută o singură dată. Poate să conțină și declararea unor variabile. În acest caz, variabilele vor exista numai în instrucțiunea for.

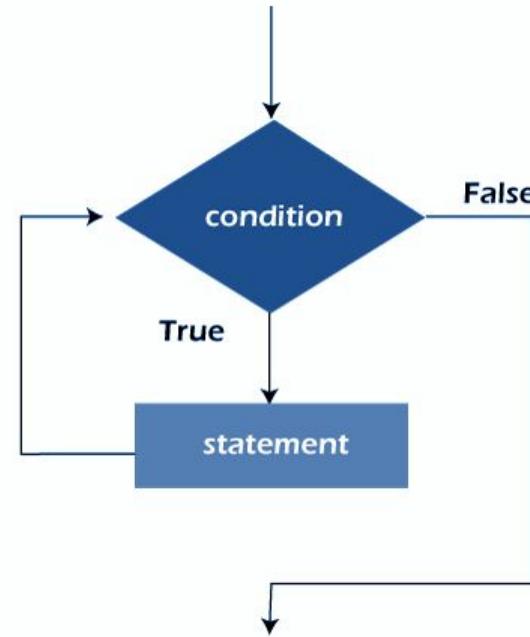
Elemente lipsa	Exemplu cod java	Output
Initializare	<pre>int i=0; for(;i<5;i+=2){ System.out.println(i); }</pre>	0 2 4
Expresie booleana	<pre>for(int i=1;i++){ System.out.println(i); if(i==2){ break; } }</pre>	1 2
Pas	<pre>for(int i=1;i<3;){ i++; System.out.println(i); }</pre>	1 2
Initializare Expresie booleana	<pre>int i=3; for(;i-;{ if(i<0){ break; } System.out.println(i); }</pre>	3 2 1 0
Initializare Pas	<pre>int i=-1; for(;i<3;{ i+=2; System.out.println(i); }</pre>	1 3
Expresie booleana Pas	<pre>for(int i=1;){ System.out.println(i); if(i>3){ break; } i=i*2; }</pre>	1 2 4
Initializare Expresie booleana Pas	<pre>for();{ }</pre>	loop infinit

For each

```
public class DayOfWeek {  
    public static void main(String[] args) {  
        System.out.println("There are 7 weekdays:");  
        int i=1;  
        for(String day:new String[]{"Monday","Tuesday","Wednesday","Thursday",  
"Friday", "Saturday","Sunday"})  
        {  
            System.out.println(i++ +" "+day);  
        }  
    }  
}
```

Sintaxa

- **while (Expresie) {Instructiune:}**



Mod de execuție

- Se evaluatează Expresie
- Dacă Expresie este nenulă
 - Se execută Instrucțiune;
 - Se reia pasul 1.
- Dacă Expresie este nulă, se trece la instrucțiunea de după while.

Observații

- Instrucțiune; se execută cât timp Expresie este nenulă – condiție adevărată.
- Dacă Expresie este vidă, atunci Instrucțiune; nu se execută deloc.
- Instrucțiune; poate fi orice fel de instrucțiune, dar una singură. Dacă sunt necesare mai multe instrucțiuni, se va folosi instrucțiunea compusă.
- Este necesar ca cel puțin o variabilă care apare în Expresie să-și modifice valoarea în Instrucțiune;

Altfel se obține o **bucătă infinită**.

Instrucțiunea break

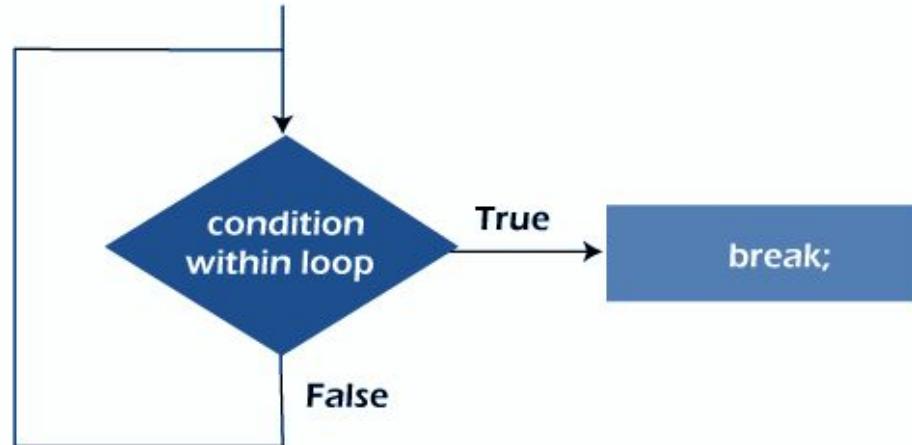
- Instrucțiunea break are sens și poate fi folosită numai în instrucțiunile switch, while, do ... while și for.

Sintaxa:

- **break;**

Mod de execuție

- Am văzut semnificația instrucțiunii break atunci când apare în instrucțiunea switch.
- Efectul instrucțiunii break când apare într-o instrucțiune repetitivă este întreruperea execuției acesteia și trecerea la instrucțiunea care urmează celei repetitive.

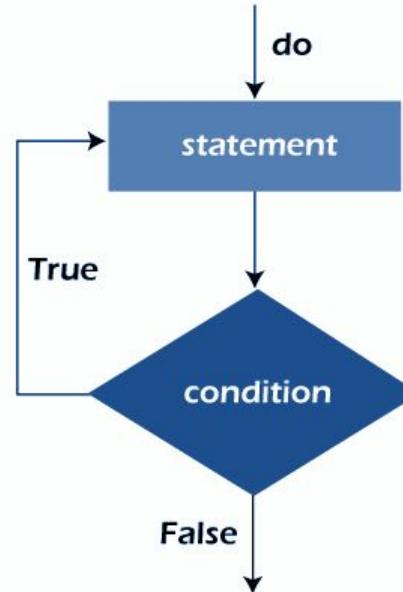


Q2. 3.What's the output of the following code?

```
String[] programmers = {"Paul", "Shreya", "Selvan", "Harry";
for (String name : programmers) {
    if (name.equals("Shreya"))
        break;
    System.out.println(name);
}
```

Instructiunea do ... while

- Sintaxa
- **do Instructiune;**
while (Expresie);



Mod de execuție

- Se execută Instrucțiune;
- Se evaluatează Expresie
- Dacă Expresie este nenulă, se reia pasul I.
- Dacă Expresie este nulă, se trece la instrucțiunea de după do ... while.

Observații

- Instrucțiune; se execută cât timp Expresie este nenulă – condiție adevărată.
- Dacă Expresie este de început vidă, Instrucțiune; se execută exact o dată. În orice situație, Instrucțiune se execută cel puțin o dată.
- Instrucțiune; poate fi orice fel de instrucțiune, dar una singură. Dacă sunt necesare mai multe instrucțiuni, se va folosi instrucțiunea compusă.
- Este necesar ca cel puțin o variabilă care apare în Expresie să-și modifice valoarea în Instrucțiune;. Altfel se obține o **bucă infinită**.

Q2.4. What's the output of the following code?

```
class Loop2 {  
  
    public static void main(String[] args) {  
  
        int i = 10;  
  
        do  
  
            while (i++ < 15)  
  
                i = i + 20;  
  
            while (i < 2);  
  
            System.out.println(i);  
  
        }  
  
    }  
  
}
```

a. 10

b. 30

Problema

Elaborează un program care genereaza aleator un număr din intervalul [1..10], cere utilizatorul să ghiceasca și afișează la ecran: Ai câștigat (în caz de coincidență) și mai încearcă o data până cu ghicește.

```
Random randNumber = new Random();  
int unknown_number = randNumber.nextInt(10) + 1;
```

```
import java.util.*;
public class NumberGenerate{
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        Random randNumber = new Random();
        int unknown_number = randNumber.nextInt(10) + 1;
        int enter_number;
        do  {
            System.out.println( "Enter unknown number [1:10] :");
            enter_number=sc.nextInt();
        } while ( enter_number != unknown_number );
        System.out.println("You win!!!\n");
    }
}
```

Instrucțiunea continue

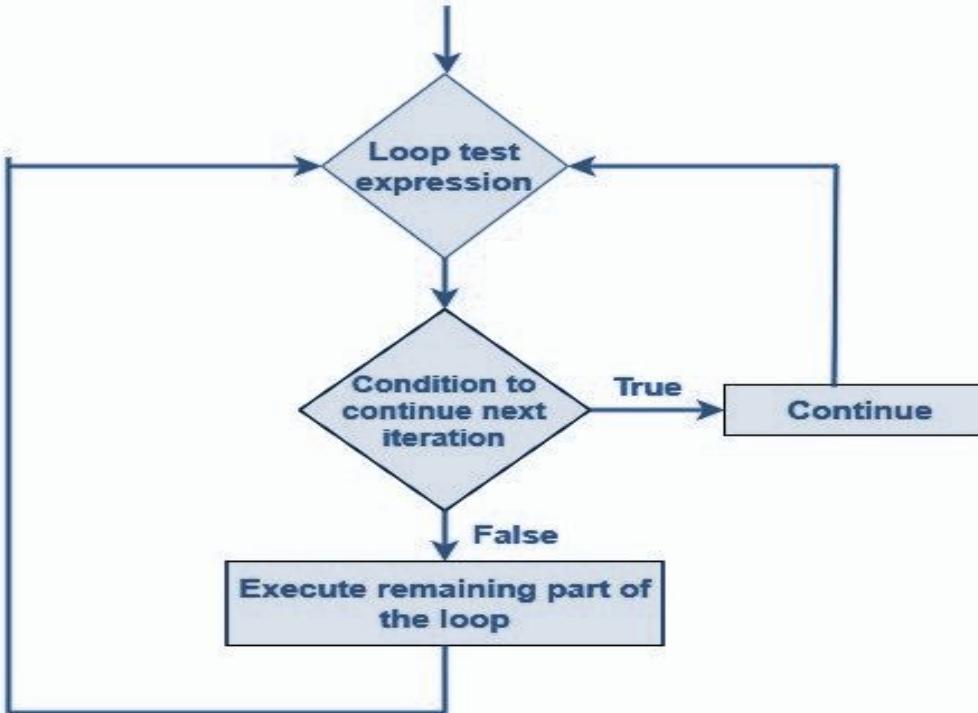
Instrucțiunea continue are sens și poate fi folosită numai în instrucțiunile while, do ... while și for.

Sintaxa:

- **continue;**

Mod de execuție

- Efectul instrucțiunii continue este ignorarea instrucțiunilor care îi urmează în corpul ciclului și revenirea la evaluarea Expresiei, în cazul lui while, do ... while, respectiv la evaluarea Expresiei_de_Condiție, în cazul lui for.



The continue Statement

break statement

```
for or while loop {  
    :  
    :  
    break;  
    :  
    :  
}
```

Takes control here

continue statement

```
for or while loop {  
    :  
    :  
    continue;  
    :  
    :  
}
```

Takes control to the
start of next iteration

Q2.5. What's the output of the following code?

```
String[] programmers = {"Paul", "Shreya", "Selvan", "Harry";
for (String name: programmers) {
if (name.equals("Shreya")) {
continue;
}
System.out.println(name);
}
```

Labeled break Statements

You can use a labeled `break` statement to exit an outer loop.

```
String[] levels = {"Outer", "Inner"};  
  
topLevel:  
for (String outer : levels) {  
    for (String inner : levels) {  
        if (inner.equals("Inner")) {  
            break topLevel;  
        }  
        System.out.print(inner + ":");  
    }  
}
```

Output: Outer:

Labeled continue Statements

You can use a labeled `continue` statement to skip an iteration of the outer loop.

```
String[] names = {"Paul", "Julia", "Mike", "John"};  
  
topLevel:  
for (String outer : names) {  
    for (String name : names) {  
        if (name.equals("Julia")) {  
            continue topLevel;  
        }  
        System.out.print(name + ":");  
    }  
}
```

Output: Paul
Paul
Paul
Paul