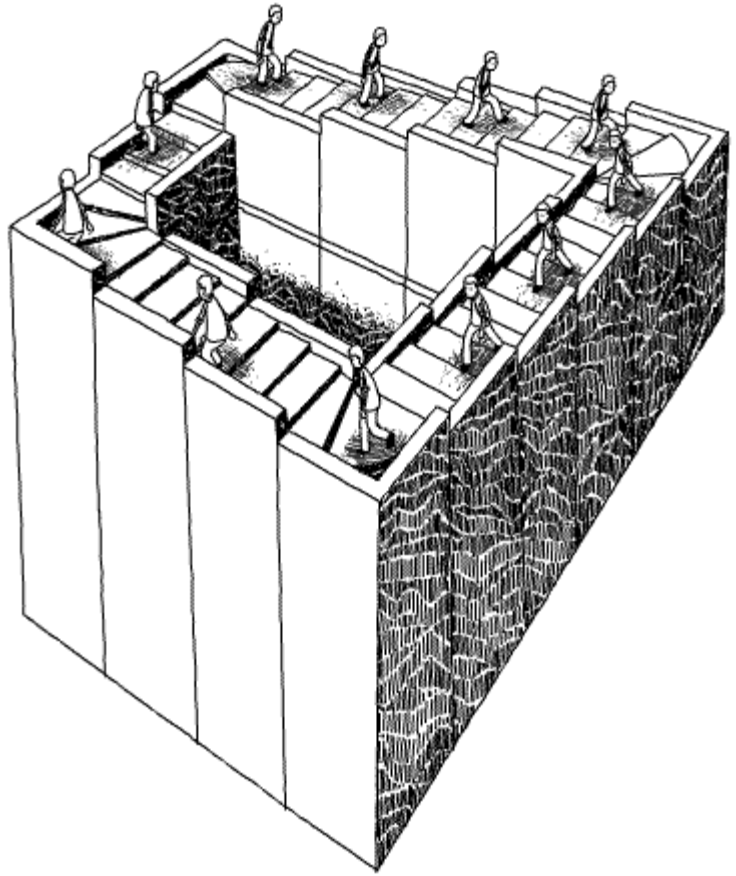
# Structures répétitives



# Structures répétitives

- Une structure répétitive permet de répéter un traitement, un certain nombre de fois.
  - Le nombre de répétitions peut-être connu à l'avance ou inconnu.
  - On dit aussi structures itératives, ou plus communément boucles

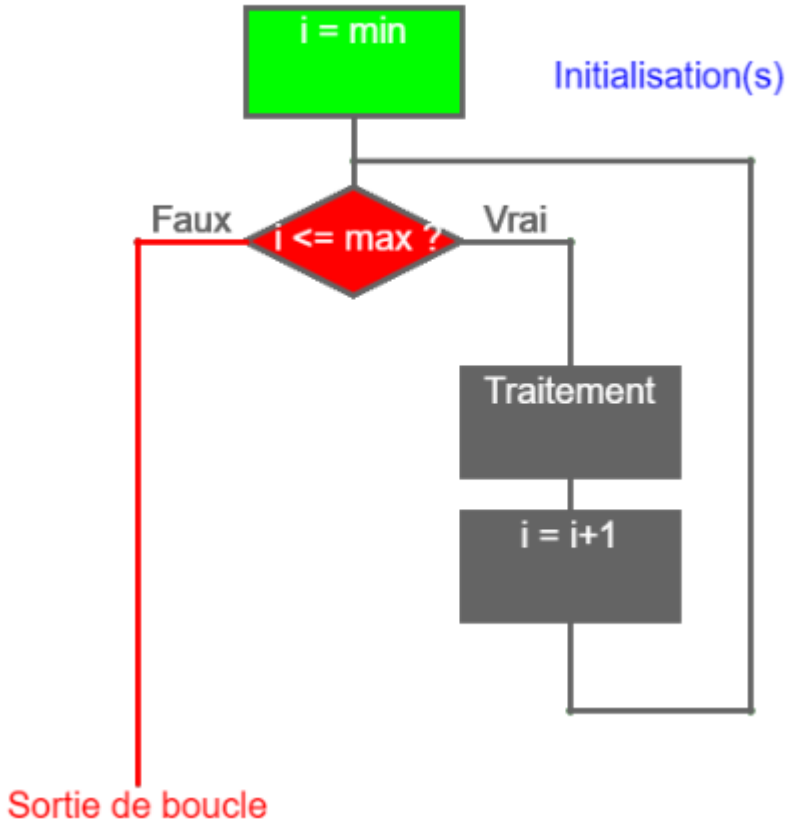
- Trois structures répétitives sont à notre disposition :

  - POUR … DE …À…
  - TANT QUE ….. FAIRE
  - RÉPÉTER …. TANT QUE

- Chacune de ces structures présente des caractéristiques et usages propres.

# Structure répétitive POUR

Employée lorsque **le nombre d'itérations est connu à l'avance…**

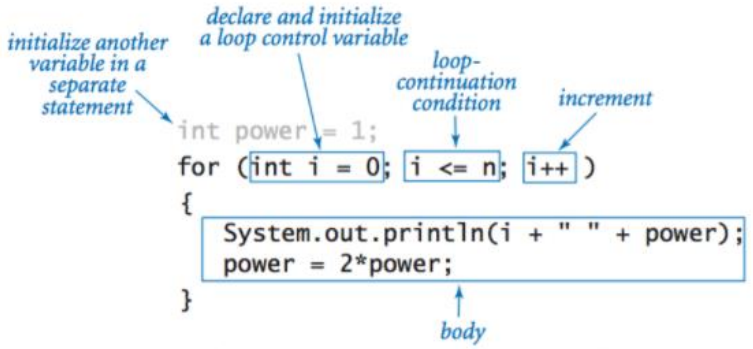



```
POUR i DE min À max FAIRE
        {traitement}
FIN_POUR
```

1) La variable **i** est souvent appelée **compteur de boucle**.
2) On peut utiliser la valeur du compteur au sein du *traitement*.
3) **Ne pas modifier le compteur au cours du *traitement*.**
4) Si max < min, on ne rentre pas dans la boucle

# Pour en Java (for)

**Anatomy of a for loop.**




•L'instruction d'initialisation "int i=1" comprend la déclaration du compteur.

## Syntaxe

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

**L'instruction 1** est exécutée (une fois) avant l'exécution du bloc de code.

**L'instruction 2** définit la condition d'exécution du bloc de code.

**L'instruction 3** est exécutée (à chaque fois) après l'exécution du bloc de code.

L'exemple ci-dessous imprimera les nombres de 0 à 4 :

## Exemple

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

# Structure répétitive POUR

## Variantes

- Le compteur peut être mis à jour en décroissant

```
POUR i DE max À min EN DESCENDANT FAIRE
        {traitement}
FIN_POUR
```

- Le compteur peut être incrémenté ou décrémenté d'une valeur autre que 1

```
POUR i DE min À max PAR PAS DE 2 FAIRE
        {traitement}
FIN_POUR
```

- Les deux variantes peuvent être combinées

```
POUR i DE max À min EN DESCENDANT PAR PAS DE 3 FAIRE
        {traitement}
FIN_POUR
```

- Le *pas* d'une boucle *for* est toujours positif ; c'est le sens de parcours qui doit varier (croissant/décroissant)

# Structure répétitive POUR

## Exemples de variantes

### Compte à rebours

```
variables
        nbSecondes entier
Début
        lire(nbSecondes)
        POUR i DE nbSecondes À 1 EN DESCENDANT FAIRE
                écrire(i)
                attendre une seconde
        FIN_POUR
        écrire("BOUM !")
Fin
```

### Pairs/impairs

```
Constantes
        N_MAX_PAIR entier =100
Début
        POUR i DE 0 À N_MAX_PAIR PAR PAS DE 2 FAIRE
                écrire(i)                                       // les pairs
        FIN_POUR

        POUR i DE N_MAX_PAIR-1 À 1 EN DESCENDANT PAR PAS DE 2 FAIRE
                écrire(i)                                       // les impairs
        FIN_POUR
Fin
```

# Structure répétitive POUR

En Java, la structure générale d'une boucle **for** est la suivante :

```
for ( instructions d'initialisation(s) ; condition de continuation ; instructions de mise à jour ){
        // traitement
}
```

- Les instructions d'initialisation et de mise à jour peuvent être multiples ; elles sont alors séparées par des **virgules**.

```
for(int i =0, int j=10 ; i<2 && j>=3 ; i++, j--){
        // traitement
}
```

- Les instructions de mise à jour sont effectuées **après** chaque *traitement*.

```
for(int i =0 ; i<2 ; i++){
        //traitement
}
System.out.println(i) ;
```

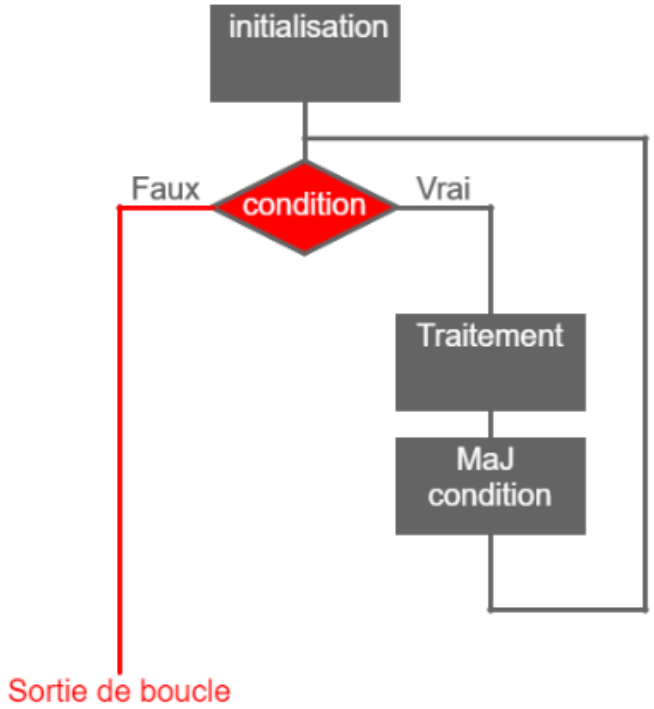
- Si la condition de continuation est fausse, on sort de la boucle.

```
for(int i =0 ; i==2 ; i++){
        System.out.println(i);  // ne sera jamais effectué
}
```

1) Les **instructions d'initialisation** et de mise à jour **peuvent être multiples**.
2) **Si la condition est fausse à la première évaluation, le traitement n'est jamais effectué.**

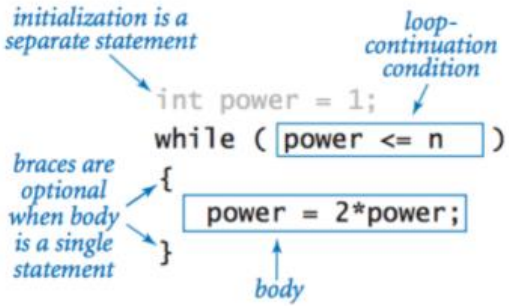
# Structure répétitive TANT QUE

Employée lorsque **le nombre d'itérations est inconnu.**




```
{initialisation(s)}
TANT QUE ( condition == vrai ) FAIRE
   {traitement}
   {mise à jour condition}
FIN_TANT_QUE
```

**Anatomy of a while loop.**




- **Il faut se donner une chance de sortir de la boucle**

⇒ **mise à jour les variables de la condition.**

# For vs While

## For

```java
for (int i=min; i <= max ; i++){
        //traitement
}
```

## While

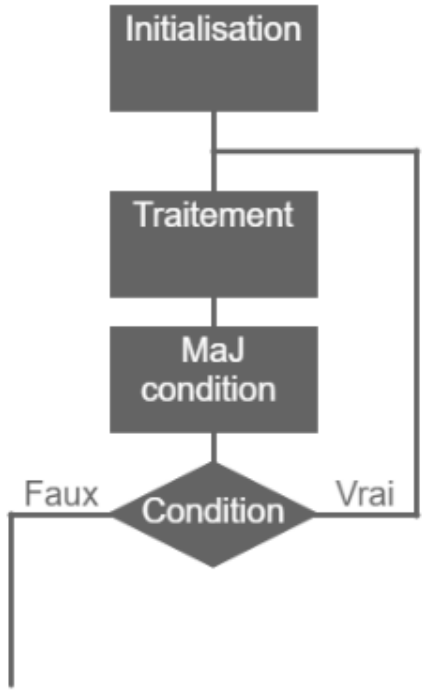
```java
i = min;
while (i <= max){
        //traitement
        i++;
}
```

L'intérêt du *for* par rapport au *while* est la déclaration locale du compteur et la gestion de ses mises à jours : Le corps de la boucle ne contient que le traitement *utile*.

# Structure répétitive RÉPÉTER

Employée lorsque le nombre d'itérations est inconnu.

```
{initialisation(s)}
RÉPÉTER
   {traitement}
   {mise à jour condition}
TANT QUE ( condition == vrai )
```




- Il faut se donner une chance de sortir de la boucle

  ⇒ modifier au moins une des variables de la condition.

# Structure répétitive RÉPÉTER

## Exemple 3 : lecture de données

### Pseudocode

```
Idée
        "On attend au clavier un nombre entre 1 et 100."
variables
        n entier
Début
        RÉPÉTER
                écrire("Entrez un nombre entre 1 et 100 : ")
                lire(n)
        TANT QUE !((i ≥ 1) && (i ≤ 100))
        écrire("OK")
Fin
```

### Java

```
int n;
do{
        System.out.println("Entrez un nombre entre 1 et 100 : ");
        n = input.nextInt();
} while (!(i <= 100) && (i >= 1));
System.out.println("OK");
```

# Structures répétitives

## Imbrication

- On peut imbriquer des boucles quels qu'en soient les types.
- L'imbrication permet classiquement de *démultiplier* le nombre d'instructions effectuées.

**Exemple 4 : valeur moyenne d'une image en niveau de gris.**

```
Constantes
        I_DIM entier // hauteur
        J_DIM entier // largeur
variables
        somme entier
Début
somme := 0
POUR i DE 0 à I_DIM-1 FAIRE
        POUR j DE 0 à J_DIM-1 FAIRE
                somme := somme + pixel(i,j)  // I_DIM x J_DIM pixels visités
        FIN_POUR
FIN_POUR
écrire("Moyenne = " + (réel)somme/(I_DIM*J_DIM))
```

# Structures répétitives

## Instructions de branchement : `break`

L'instruction **break** provoque la sortie immédiate de la boucle en cours.

```
int max = 100;
for(int i=0; i< max; i++){
        if (i == 50) break ;
}
```

Boucles imbriquées

```
final int I_DIM = 512
final int J_DIM = 512

double somme = 0;
boolean erreur = false;

for(int i=0; i< I_DIM; i++)
        for(int j=0; j< J_DIM; j++){
                if( pixel(i,j) <0) {
                        erreur = true;
                        break;                  // prochaine ligne
                }
                somme += pixel(i,j);
        }
if (erreur)
        System.out.printl("Erreur");
else
        System.out.printl("Moyenne = " + (somme/(I_DIM*J_DIM));
```

# Structures répétitives

## Instructions de branchement : `break`

Boucles imbriquées

```
final int I_DIM = 512
final int J_DIM = 512

double somme = 0;
boolean erreur = false;

outerLoop:
for(int i=0; i< I_DIM; i++)
        for(int j=0; j< J_DIM; j++){
                if( pixel(i,j) <0) {
                        erreur = true;
                        break outerLoop; // sort des boucles imbriquées
                }
                somme += pixel(i,j);
        }
if (erreur)
        System.out.printl("Erreur");
else
        System.out.printl("Moyenne = " + (somme/(I_DIM*J_DIM));
```

# Structures répétitives

## Instructions de branchement : `continue`

L'instruction **continue** provoque l'arrêt immédiat de l'itération en cours et le branchement à l'itération suivante.

```
int max = 100;
for(int i=0; i< max; i++){
        if (i == 13) continue ;  // superstitieux
}
```

- Dans le cas des boucles *for*, la mise à jour de la condition est assurée.
- Dans le cas des boucles *while* et *do-while*, il faut s'assurer que la mise à jour de la condition est toujours effective.

# Structures répétitives

Que va afficher le programme suivant ?


```
Programme afficheur
variables
        i entier := 3
Début
TANT QUE i < 6 FAIRE
 | écrire(i)
 | i := i+1
FIN_TANT_QUE
Fin
```

A. 3 4 5
B. 1 2 3
C. 3 4 5 6

# Structures répétitives

Que va afficher le programme suivant ?


```
Programme afficheur
variables
        i entier := 0
Début
TANT QUE i < 0 FAIRE
 | écrire(i)
 | i := i+1
FIN_TANT_QUE
Fin
```

A. rien
B. 0
C. il va produire une erreur !

# Structures répétitives

Que va afficher le programme suivant ?


```
Programme afficheur
variables
        x réel := 3
        i entier := 0
Début
TANT QUE i < 3 FAIRE
 | x := x+1
 | i := i+1
FIN_TANT_QUE
écrire(x)
Fin
```

A. 3
B. 7
C. 6
D. 4

# Exercice n° 1:

Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

# Correction

```
Variables
 n  Entier

Début

    Ecrire ("Entrez un nombre entre 1 et 3: ")
    RÉPÉTER
            Lire (n)
            SI ((n < 1) ou (n > 3)) Alors
                Ecrire ("Saisie erronée. Recommencez")
            FIN_SI
      TANT QUE (n < 1 ou n > 3)
     Ecrire("n =" +n)
FIN
```

```
Variables
 n  Entier

Début

      n := 0
      Ecrire ("Entrez un nombre entre 1 et 3: " )
      Tant Que (n < 1 ou n > 3) FAIRE
         Lire (n)
         SI ((n < 1) ou (n> 3)) Alors
             Ecrire ("Saisie erronée. Recommencez")
         Fin_SI
       Fin_Tant_Que

Fin
```

# Correction

```java
import java.util.*;
class SaisieDoWhile{

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        int N;
        System.out.println("Entrez un nombre entre 1 et 3: ");
        do{
            N=input.nextInt();
            if ((N<1) || (N>3))
                System.out.println("Saisie erronée. Recommencez. \n Entrez un nombre entre 1 et 3: ");
        } while((N<1) || (N>3));
        System.out.println("N: "+N);
    }
}
```

```java
import java.util.*;
class SaisieWhile{

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){

        int N=0;
        System.out.println("Entrez un nombre entre 1 et 3: ");
        while((N<1) || (N>3)){
            N=input.nextInt();
            if ((N<1) || (N>3))
                System.out.println("Saisie erronée. Recommencez. \n Entrez un nombre entre 1 et 3: ");
        }
        System.out.println("N: "+N);
    }
}
```

# Exercice n° 2:

Ecrire un algorithme qui demande un nombre compris entre une borne inférieur et une borne supérieur, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à la borne supérieur, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à la borne inférieur.

# Correction

**Variables:**

| | |
|---|---|
| n | **Entier** |
| bornInf | **Entier** |
| bornSup | **Entier** |

**Début**

**Ecrire** ("Entrez la borne inférieur?: ")
**Lire**(bornInf)
**Ecrire** ("Entrez la borne supérieur?: ")
**Lire**(bornSup)
**Ecrire** ("Entrez un nombre entre "+ bornInf+ " et "+ bornSup +" : ")

**RÉPÉTER**
**Lire** (n)
**SI** (n < bornInf ) **ALORS**
**Ecrire** (" Entrez un nombre plus grand !:" )
**SINON SI** (n > bornSup) **ALORS**
**Ecrire** (" Entrez un nombre plus petit !:" )
**FIN_SI**
**Tant Que** ((n < bornInf) ou (n > bornSup))

**Fin**

# Avec do-while

```java
import java.util.*;
class SaisieDoWhileComprisEntre {

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        int N;
        int bornInf;
        int bornSup;
        System.out.println("Entrez la borne inférieure :");
        bornInf=input.nextInt();
        System.out.println("Entrez la borne supérieure:");
        bornSup=input.nextInt();

        System.out.println("Entrez un nombre entre "+ bornInf+ " et "+ bornSup +" : ");
        do{
            N=input.nextInt();
            if (N<bornInf)
                System.out.println("Saisie erronée. Recommencez. \n Entrez un nombre plus grand !: ");
            else if (N>bornSup)
                System.out.println("Saisie erronée. Recommencez. \n Entrez un nombre plus petit !: ");

        } while((N<bornInf) || (N>bornSup));
        System.out.println("N: "+N);
    }
}
```

# Avec while()

```
import java.util.*;
class SaisieWhileComprisEntre {

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        int N;
        int bornInf;
        int bornSup;
        System.out.println("Entrez la borne inférieure :");
        bornInf=input.nextInt();
        System.out.println("Entrez la borne supérieure:");
        bornSup=input.nextInt();

        N=bornInf-1;

        System.out.println("Entrez un nombre entre "+ bornInf+ " et "+ bornSup +" : ");

        while((N<bornInf) || (N>bornSup)){
            N=input.nextInt();
            if (N<bornInf)
                System.out.println("Saisie erronée. Recommencez. \n Entrez un nombre plus grand !: ");
            else if (N>bornSup)
                System.out.println("Saisie erronée. Recommencez. \n Entrez un nombre plus petit !: ");

        }
        System.out.println("N: "+N);
    }
}
```

# Exercice n° 3:

a) Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les **dix nombres suivants**, en utilisant l'instruction **POUR**.

Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

b) Réécrire l'algorithme précédent, en utilisant cette fois l'instruction **TANT QUE**

# Correction

```
Variables
        n Entier
        i  Entier

Début
        Ecrire ("Entrez un nombre : ")
        Lire (n)
        Ecrire ("Les 10 nombres suivants sont :  ")
        Pour i  de 1 à 10 FAIRE
                Ecrire ("  " + (n + i))
         FIN_POUR
Fin
```

```
Variables
        n Entier
        i  Entier

Début

        Ecrire ("Entrez un nombre : ")
        Lire (n)
        i:= 1
        Ecrire ("Les 10 nombres suivants sont :  ")
        Tant Que (i < =10) FAIRE
           Ecrire ("  " + (n + i))
            i := i + 1
        Fin_Tant_Que

Fin
```

# Correction (JAVA)

```java
import java.util.*;
class AfficheDixNombreSuivantsPour {

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        int N;
        System.out.print("Entrez un nombre :");
        N=input.nextInt();
        System.out.println("Les 10 nombres suivants sont: ");
        for(int i=1;i<=10;i++){
        System.out.print("  " + (N + i));
        }
    }
}
```

```java
import java.util.*;
class AfficheDixNombreSuivantsWhile {

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        int N;
        int i=1;
        System.out.print("Entrez un nombre :");
        N=input.nextInt();
        System.out.println("Les 10 nombres suivants sont: ");
        while(i<=10){
            System.out.print("  " + (N + i));
            i++;
        }
    }
}
```

# Exercice n° 4:

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

```
Table de 7 :
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 7
```

# Correction

**Variables**

n **Entier**

i **Entier**

**Debut**

**Ecrire** ("Entrez un nombre : " )
**Lire** (n)
**Ecrire** ("La table de multiplication de ce nombre est : ")
**Pour** i **de** 1 **à** 10 **FAIRE**
**Ecrire** (n+ " x "+ i+ " = "+ n*i)
**FIN_POUR**

**Fin**

```java
import java.util.*;
class TableDeMultiplicationDeNombre {

    public static Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        int N;
        System.out.print("Entrez un nombre :");
        N=input.nextInt();
        for(int i=1;i<=10;i++)
            System.out.println(N+ " x "+ i+ " =  "+ N*i);
    }
}
```