



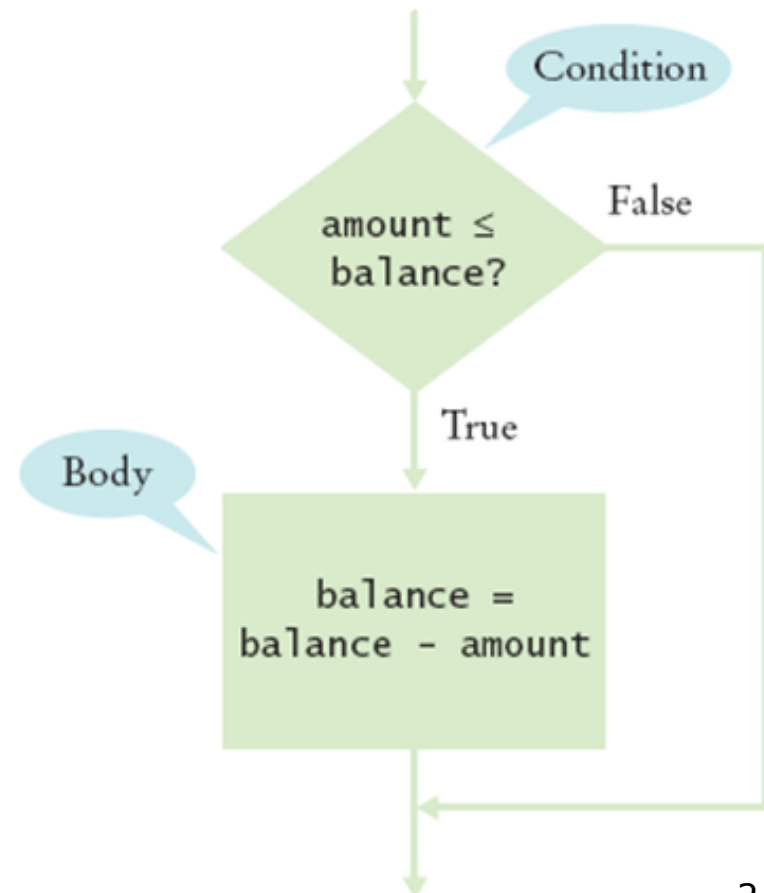
Decisioni

Istruzione if

```
if (amount <= balance)
    balance = balance - amount;
```

Sintassi:

```
if (condizione)
    istruzione
```

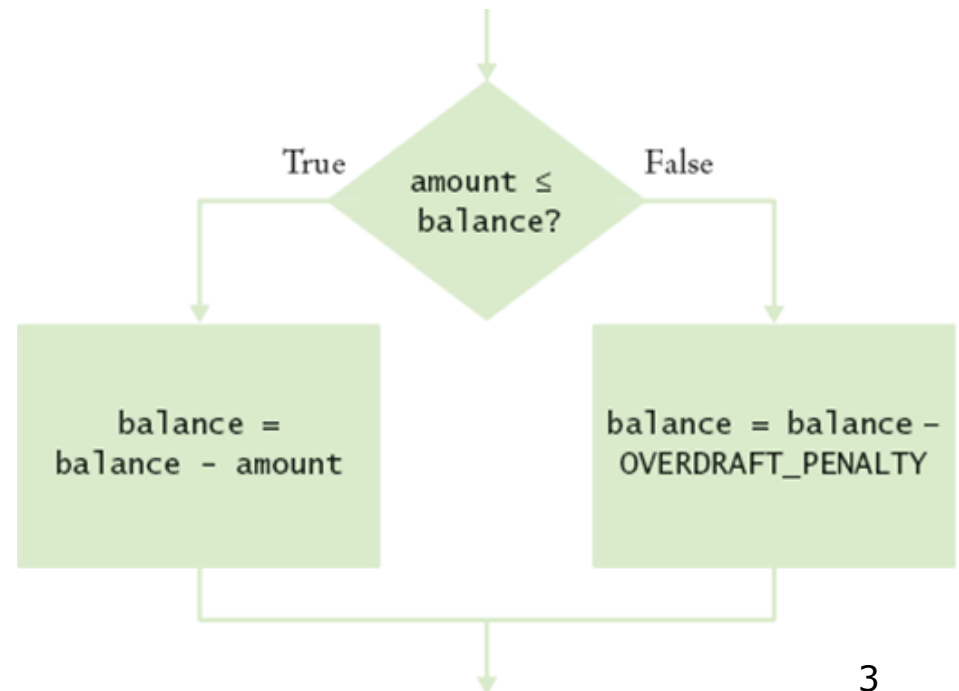


Istruzione if/else

```
if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - OVERDRAFT_PENALTY;
```

Sintassi:

```
if (condizione)
    istruzione
else
    istruzione
```





Blocco di istruzioni

```
{  
    istruzione1  
    istruzione2  
    . . .  
}
```

Obiettivo:

Raggruppare più istruzioni
per formare un'unica
istruzione

Esempio:

```
if (amount <= balance)  
{  
    double newBalance =  
        balance - amount;  
    balance = newBalance;  
}
```

Se la condizione dell'**if**
è verificata vengono
eseguiti tutti gli statement
all'interno del blocco



Tipi di istruzioni

- **Semplice**

```
balance = balance - amount;
```

- **Composto**

```
if (balance >= amount)
    balance = balance - amount;
```

- **Blocco di istruzioni**

```
{
    double newBalance = balance - amount;
    balance = newBalance;
}
```



Esempio:

Retribuzione dei dipendenti

○ **Presentazione del problema**

- Modellare un sistema di retribuzione per dipendenti che sono pagati con una tariffa oraria. Il sistema deve riuscire a calcolare la retribuzione di un dipendente sulla base della tariffa oraria e delle ore di lavoro effettuate e deve stampare il nome, le ore e la paga calcolata. I dipendenti che lavorano più di 40 ore ricevono una somma per gli straordinari, pagati una volta e mezzo la loro tariffa salariale normale. Se un dipendente ha 30 o più ore di straordinario nelle ultime due settimane viene emesso un messaggio d'avviso



Retribuzione

○ Scenario d' esempio

Enter employee name: **Gerald Weiss**

Enter employee rate/hour: **20**

Enter Gerald Weiss' s hours for week 1: **30**

Gerald Weiss earned \$600 for week 1

Enter Gerald Weiss' s hours for week 2: **50**

Gerald Weiss earned \$1100 for week 2

Enter Gerald Weiss's hours for week 3: **60**

Gerald Weiss earned \$1400 for week 3

*** Gerald Weiss has worked 30 hours of overtime in the last two weeks.



Retribuzione

- Oggetti primari
 - Termini chiave: dipendente, ore, nome, retribuzione oraria ...
 - **DIPENDENTE**, gli altri descrivono attributi del dipendente

```
class Employee {  
  ...  
}
```




Retribuzione

- Comportamento desiderato
 - Creare oggetti di tipo Employee (costruttore)
 - **Employee**
 - Calcolare la paga
 - **calcPay**
 - Interrogare un oggetto Employee per conoscere il nome
 - **getName**



Retribuzione

- Costruttore
 - È necessario specificare il nome e la paga oraria
 - Le ore lavorate cambiano di settimana in settimana
- Il calcolo della paga richiede la conoscenza del numero di ore lavorate
- Il ritrovamento del nome restituisce una stringa

```
class Employee {  
    public Employee(String name, int rate) {...}  
    public int calcPay(int hours) {...}  
    public String getName() {...}  
    ...  
}
```



Retribuzione

○ Variabili d'istanza

- È necessario conservare il nome di un dipendente e la sua paga oraria (getName, calcPay)
- È anche necessario memorizzare le ore di straordinario dell'ultima settimana

```
class Employee {  
    ...  
    // Methods  
    // Instance variables  
    private String name;  
    private int rate;  
    private int lastWeeksOvertime;  
}
```



Retribuzione

- Costruttore

- Inizializza le variabili di stato con i valori degli argomenti

```
public Employee(String name, int rate) {  
    this.name = name;  
    this.rate = rate;  
    this.lastWeeksOvertime = 0;  
}
```



Retribuzione

- getName
 - Restituisce il nome del dipendente in una String

```
public String getName() {  
    return this.name;  
}
```



Retribuzione

○ calcPay

- Prima si verifica se si è superato il limite delle 40 ore per verificare se c'è straordinario e poi si calcola la paga
- Il metodo calcola inoltre il numero di ore di straordinario

```
int pay, currentOvertime;
if (hours <= 40) {
    pay = hours * rate;
    currentOvertime = 0;
} else {
    pay = 40*rate + (hours-40)*(rate+rate/2);
    currentOvertime = hours - 40;
}
```



Retribuzione

- calcPay

- Per la gestione del messaggio d'allarme si addiziona il numero di ore di straordinario della settimana corrente a quello della settimana passata e lo si confronta con 30

```
if (currentOvertime + lastWeeksOvertime >= 30)
    System.out.print(name + " has worked 30" +
        " or more hours overtime");
```



Retribuzione

- calcPay
 - L'ultimo atto è la memorizzazione del numero di ore di straordinario e la restituzione della paga

```
lastWeeksOvertime = currentOvertime;  
return pay;
```




calcPay

```
public int calcPay(int hours) {  
    int pay, currentOvertime;  
    if (hours <= 40) {  
        pay = hours * rate;  
        currentOvertime = 0;  
    } else {  
        pay = 40*rate+(hours-40)*(rate+rate/2);  
        currentOvertime = hours - 40;  
    }  
    if (currentOvertime + lastWeeksOvertime >= 30)  
        System.out.print(name + " has worked " +  
            " 30 or more hours overtime");  
    lastWeeksOvertime = currentOvertime;  
    return pay;  
}
```



Retribuzione

- Programma d' esempio

```
class Payroll {  
    public static void main(String a[]) {  
        Employee e;  
        e = new Employee("Rudy Crew", 10);  
        int pay;  
        pay = e.calcPay(30);  
        System.out.print(e.getName());  
        System.out.print(" earned ");  
        System.out.println(pay);  
    }  
}
```

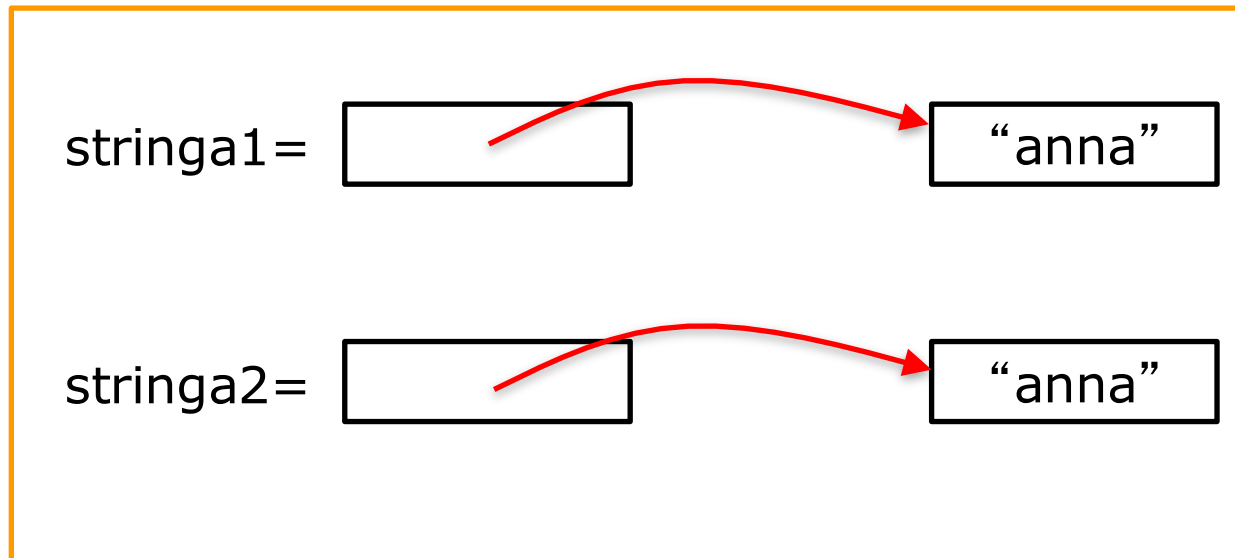
Confronto tra stringhe (1)

- Non bisogna usare `==` per confrontare due stringhe

```
if (stringa1 == stringa2)
```

```
// Testa se stringa1
```

```
// e stringa2 si riferiscono alla stessa stringa
```



`stringa1 == stringa2`
restituisce **false**



Confronto tra stringhe (2)

```
String stringa1="Anna";
String s = "Annamaria";
String stringa2 = s.substring(0,4);
if (stringa1 == stringa2)
    System.out.println("stringhe uguali");
else
    System.out.println("stringhe diverse");

//il programma stampa "stringhe diverse"
```



Confronto tra stringhe (3)

- Per confrontare due stringhe bisogna usare il metodo **equals** di String:

```
if (stringa1.equals(stringa2))  
// Testa se le stringhe a cui fanno riferimento  
// stringa1 e stringa2 sono uguali
```

- Se il confronto non deve tenere conto delle maiuscole/minuscole si usa il metodo **equalsIgnoreCase**

```
if (stringa1.equalsIgnoreCase("ANNA"))  
//il test restituisce true
```

- Vale per gli oggetti in generale
 - E' opportuno che forniate i vostri oggetti di un metodo **equals**



Confronto tra stringhe (4)

```
String stringa1="Anna";
String s = "Annamaria";
String stringa2 = s.substring(0,4);
if (stringa1.equals(stringa2))
    System.out.println("stringhe uguali");
else
    System.out.println("stringhe diverse");

//il programma stampa "stringhe uguali"
```



Confronto tra stringhe (5)

- Java crea un solo oggetto stringa per ogni stringa costante

```
String stringa1 = "Marco";
```

```
String stringa2 = "Marco";
```

```
.....
```

```
if (stringa1 == stringa2)
```

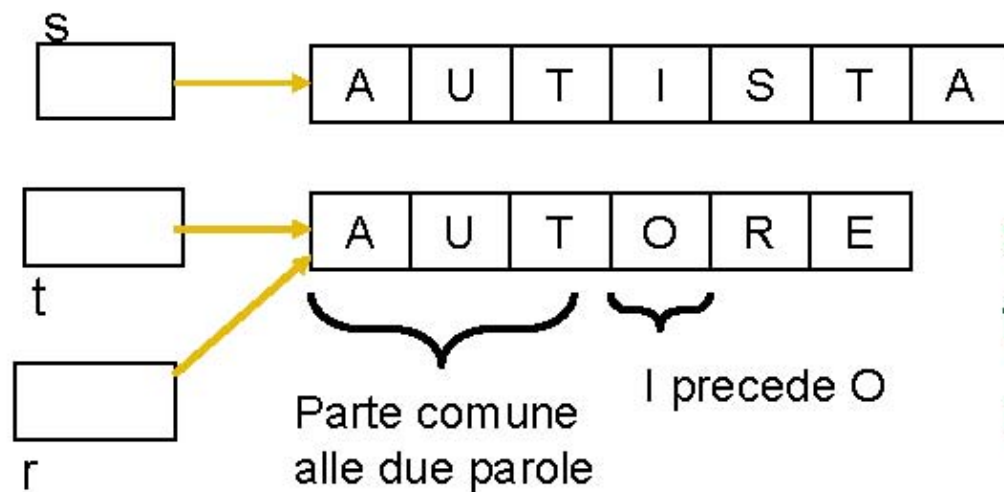
```
    //condizione vera
```



Ordine lessicografico (1)

- Si usa il metodo `compareTo` della classe `String`
 - Es.: `s.compareTo(t) < 0`
se la stringa `s` precede la stringa `t` nel dizionario
- Le lettere maiuscole precedono le minuscole
- I numeri precedono le lettere
- Il carattere spazio precede tutti gli altri caratteri

Confronto lessicografico (2)



`s.compareTo(t)` è < 0

`t.compareTo(s)` è > 0

`r.compareTo(t)` è 0



Confronto di oggetti (1)

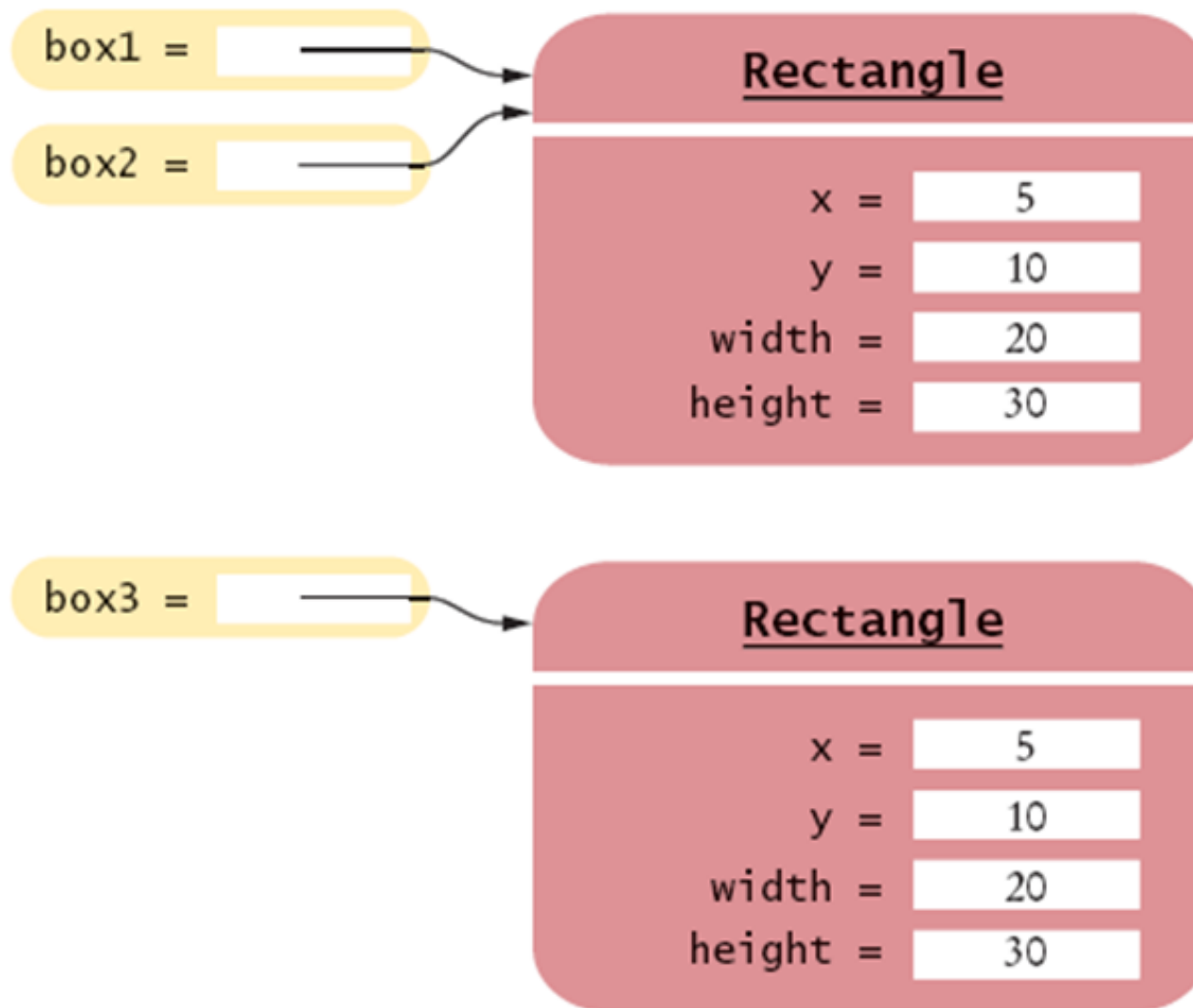
- `==` verifica se due riferimenti puntano allo stesso oggetto
- `equals` testa se due oggetti hanno contenuto identico

```
Rectangle box1 = new Rectangle(5, 10, 20, 30);  
Rectangle box3 = new Rectangle(5, 10, 20, 30);  
Rectangle box2 = box1;
```

```
if (box1 == box3)  
    System.out.println("box1 e box3 si riferiscono allo stesso rettangolo");  
if (box1.equals(box3))  
    System.out.println("box1 e box3 si riferiscono a rettangoli uguali");  
if (box2 == box1)  
    System.out.println("box2 e box1 si riferiscono allo stesso rettangolo");
```

- **Stampa:**
box1 e box3 si riferiscono a rettangoli uguali
box2 e box1 si riferiscono allo stesso rettangolo

Confronto di oggetti (2)





Il metodo equals

- Quando si definisce una nuova classe è opportuno definire il metodo **equals** che funziona per gli oggetti di quella classe
- Se non viene definito viene usato il metodo **equals** della classe **java.lang.Object** che però confronta gli indirizzi e non i contenuti degli oggetti



Il metodo `equals` per la classe `Name`

```
public boolean equals(Name n) {  
    return (first.equals(n.first) &&  
            last.equals(n.last) &&  
            title.equals(n.title) )  
}
```



Il riferimento `null`

- Il riferimento `null` non si riferisce ad alcun oggetto
- Per verificare se un riferimento è `null` si usa l'operatore `==`
 - Es.: `if (account == null) . . .`
- Occasionalmente vorremmo restituire un'indicazione che specifichi che non è possibile restituire alcun oggetto
 - Per esempio, se non ci sono dati in un file e invochiamo un metodo `read` non possiamo costruire alcun oggetto
 - In tale situazione usiamo il valore `null`



Verifica di fine input

- Un metodo di lettura può restituire **null** per indicare che non ci sono dati in un file
- Il metodo **read** per la classe **Name**

```
public static Name read(Scanner s) {  
    String first, last;  
    if (!s.hasNext()) return null;  
    first = s.next();  
    last = s.next();  
    return new Name(first, last);  
}
```



Alternative multiple

```
if (condizione1)  
    istruzione1;  
else if (condizione2)  
    istruzione2;  
else if (condizione3)  
    istruzione3;  
else  
    istruzione4;
```

- Viene eseguita lo statement associato alla prima condizione vera
- Se nessuna condizione è vera allora viene eseguito *istruzione4*
- Altra possibilità: **switch**

File Earthquake.java

```
// Una classe che definisce gli effetti di un terremoto.
public class Earthquake
{
    //costruttore
    public Earthquake(double magnitude)
    {    richter = magnitude;
    }

    // restituisce la descrizione dell'effetto del terremoto
    public String getDescription()
    {    String r;
        if (richter >= 8.0)
            r = "Most structures fall";
        else if (richter >= 7.0)
            r = "Many buildings destroyed";
        else if (richter >= 6.0)
            r = "Many buildings considerably damaged, some collapse";
        else if (richter >= 4.5)
            r = "Damage to poorly constructed buildings";
        else if (richter >= 3.5)
            r = "Felt by many people, no destruction";
        else if (richter >= 0)
            r = "Generally not felt by people";
        else
            r = "Negative numbers are not valid";
        return r;
    }

    //variabile di istanza
    private double richter;
}
```

File EarthquakeRunner.java

```
import java.util.Scanner;

/**
This program prints a description of an earthquake of a given magnitude.
*/
public class EarthquakeRunner
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        System.out.print("Enter a magnitude on the Richter scale: ");
        double magnitude = in.nextDouble();
        Earthquake quake = new Earthquake(magnitude);
        System.out.println(quake.getDescription());
    }
}
```



L'istruzione `switch`

```
switch (x) {  
    case value1: statement1  
        break;  
    case value2: statement2  
        break;  
    case value3: statement3  
        break;  
    default: statement4  
        break;  
}
```



L'istruzione `switch`

- La variabile **x** viene valutata e confrontata con i vari casi
 - Se il confronto con uno dei casi ha successo, viene eseguita l'istruzione corrispondente
 - In caso contrario viene eseguita l'istruzione del caso di "default"
- Alcune regole:
 - I valori dei "case" devono essere costanti o costanti letterali intere (anche String in Java 7)
 - Ad ogni caso è associata una sequenza di istruzioni (non sono necessarie le parentesi graffe)

```
case 5: System.out.print("case of 5");  
        System.out.println("more output");  
        break;
```
 - Ogni case deve terminare con un **break**



Diramazioni annidate

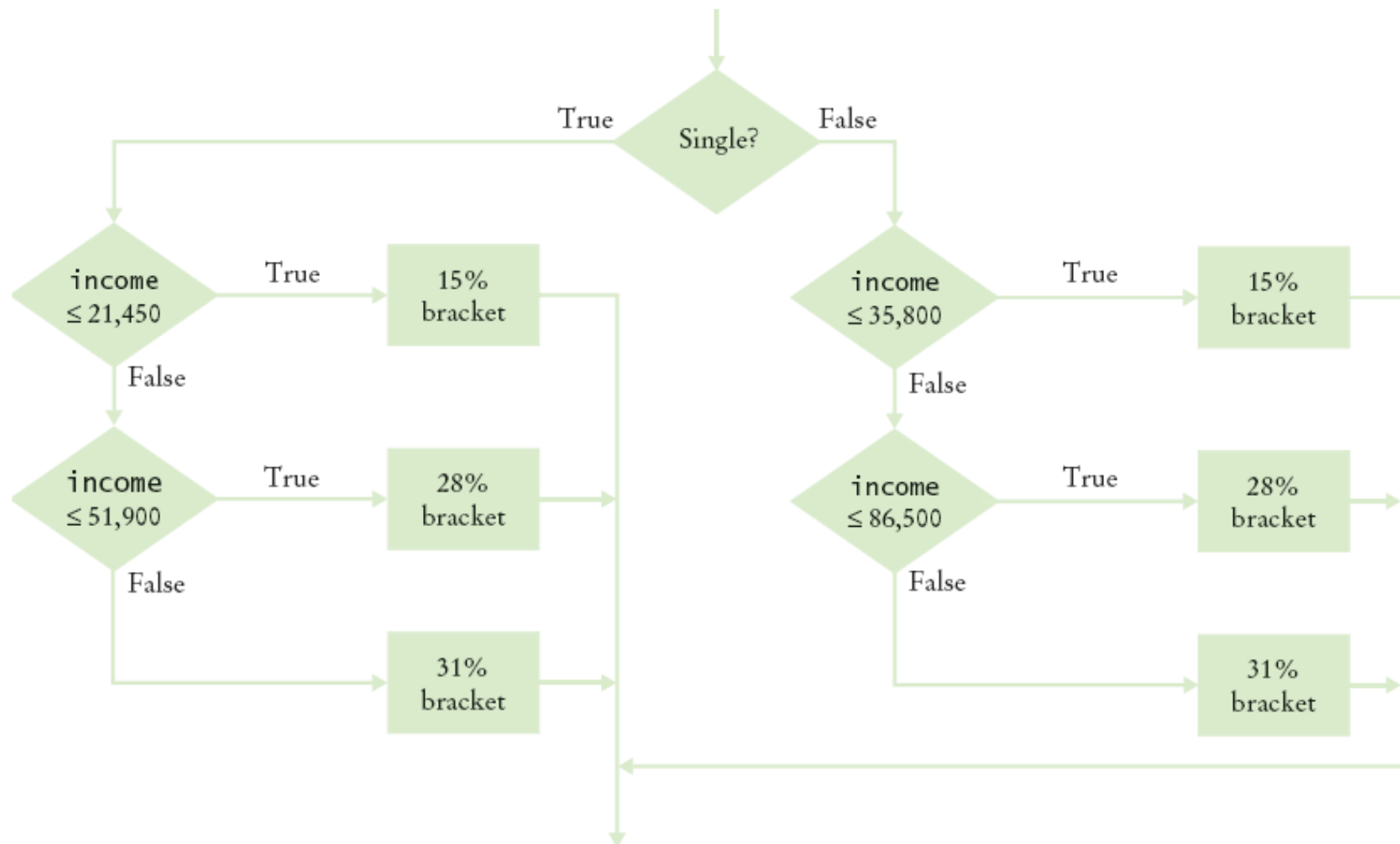
```
if (condizione1)  
{  
    if (condizione1a)  
        istruzione1a;  
    else  
        istruzione1b;  
}  
else  
    istruzione2;
```



Dichiarazione dei redditi

If your filing status is Single		If your filing status is Married	
Tax Bracket	Percentage	Tax Bracket	Percentage
\$0 . . . \$21,450	15%	0 . . . \$35,800	15%
Amount over \$21,450, up to \$51,900	28%	Amount over \$35,800, up to \$86,500	28%
Amount over \$51,900	31%	Amount over \$86,500	31%


Dichiarazione dei redditi






File TaxReturn.java

```
/** Gestione dichiarazione dei redditi
 */
public class TaxReturn
{
    /**
        Costruisce una dichiarazione dei redditi per un
        contribuente con entrate pari al valore di
        anIncome e stato civile uguale a aStatus
    */
    public TaxReturn(double anIncome, int aStatus)
    {
        income = anIncome;
        status = aStatus;
    }
}
```





```
public double getTax()
{
    double tax = 0;
    if (status == SINGLE)
    {
        if (income <= SINGLE_CUTOFF1)
            tax = RATE1 * income;
        else if (income <= SINGLE_CUTOFF2)
            tax = SINGLE_BASE2 + RATE2 *
                (income - SINGLE_CUTOFF1);
        else
            tax = SINGLE_BASE3 + RATE3 *
                (income - SINGLE_CUTOFF2);
    }
}
```



```
else
{
    if (income <= MARRIED_CUTOFF1)
        tax = RATE1 * income;
    else if (income <= MARRIED_CUTOFF2)
        tax = MARRIED_BASE2 + RATE2 *
            (income - MARRIED_CUTOFF1);
    else
        tax = MARRIED_BASE3 + RATE3 *
            (income - MARRIED_CUTOFF2);
}
return tax;
}

public static final int SINGLE = 1;
public static final int MARRIED = 2;
private static final double RATE1 = 0.15;
private static final double RATE2 = 0.28;
private static final double RATE3 = 0.31;
```



```
private static final double SINGLE_CUTOFF1 = 21450;
private static final double SINGLE_CUTOFF2 = 51900;
private static final double SINGLE_BASE2 = 3217.50;
private static final double SINGLE_BASE3 = 11743.50;
private static final double MARRIED_CUTOFF1 = 35800;
private static final double MARRIED_CUTOFF2 = 86500;
private static final double MARRIED_BASE2 = 5370;
private static final double MARRIED_BASE3 = 19566;
//variabili di istanza
private double income;
private int status;
}
```



Il problema dell' **else** sospeso (1)

```
if (a<b)
    if (b<c)
        System.out.println(b + " è compreso tra " + a + " e " + c);
else
    System.out.println(b + " è minore o uguale di " + a);
```

- **else** si lega al primo o al secondo **if**?
 - **Regola:** Java quando trova un **else** lo associa all'ultimo **if** non associato ad un **else**
 - Indentazione fuorviante nell'esempio



Il problema dell' **else** sospeso (2)

```
if (a<b)
{
    if(b<c)
        System.out.println(b + " è compreso tra " + a + " e " +c);
}
else
    System.out.println(b + " è minore o uguale di " + a);
```

- **Usiamo le parentesi graffe per forzare associazione if/else**



Espressioni booleane

- boolean è un tipo di dato primitivo che modella il comportamento di un valore di verità
 - Due possibili valori: *true* e *false*
 - La condizione di un if statement richiede un'espressione di tipo boolean
- Come per int, possiamo:
 - Dichiarare variabili boolean: *boolean b;*
 - Assegnare a queste variabili valori boolean: *b = true;*
 - Costruire espressioni boolean: *b = temperature < 32;*
 - passare booleani come argomenti
 - restituire boolean da metodi



Espressioni booleane

```
private boolean married;
```

- Inizializzato ad un valore di verità:

```
married = input.equals("M");
```

- Usato in condizioni:

```
if (married) . . . else . . .  
if (!married) . . .
```

- Chiamato anche **flag**

- E' considerato goffo scrivere

```
if (married == true) . . .
```

- Meglio usare il test semplice

```
if (married) . . .
```



Metodi predicativi

- Restituiscono un tipo booleano
- Il valore restituito dal metodo può essere utilizzato come condizione di un **if**
- Il metodo **equals** è un esempio di metodo predicativo
- La classe **Character** fornisce diversi metodi predicativi statici :

isDigit(c)

isLetter(c)

isUpperCase(c)

isLowerCase(c)

- Es. **if** (Character.isDigit(c))
 System.out.println("il carattere è una cifra");



Gli operatori booleani

- **&&** (*AND*)

- **||** (*OR*)

- **!** (*NOT*)

- Es.:

```
if (0 < amount && amount < 1000) ...
```

- La condizione dell'if è verificata se amount è compreso tra 0 e 1000

```
if (input.equals("S") || input.equals("M"))  
...
```

- La condizione dell'if è verificata se la stringa input è "S" o "M"



Precedenze di Operatori Logici

- **!** ha la priorità più alta
- **&&** viene dopo
- **||** ha la priorità più bassa
- Ovviamente è possibile sempre usare le parentesi ()

- Così,

!a && b || c

è lo stesso di

((!a) && b) || c



Domande

- Quando la seguente istruzione stampa false?
`System.out.println (x > 0 || x < 0);`

Risposta: Quando x è zero

- Riscrivere la seguente istruzione senza effettuare il confronto con `false`:

```
if (character.isDigit(ch) == false) . . .
```

Risposta: `if (!Character.isDigit(ch)) . . .`



Iterazioni



L'istruzione `while`

```
while (condition)
    istruzione
```

- Ripete l'esecuzione di istruzione fino a che la condizione resta vera

```
while (balance < targetBalance)
{
    year++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
```

Year	Balance
0	\$10,000
1	\$10,500
2	\$11,025
3	\$11,576.25
4	\$12,155.06
5	\$12,762.82

Esecuzione while

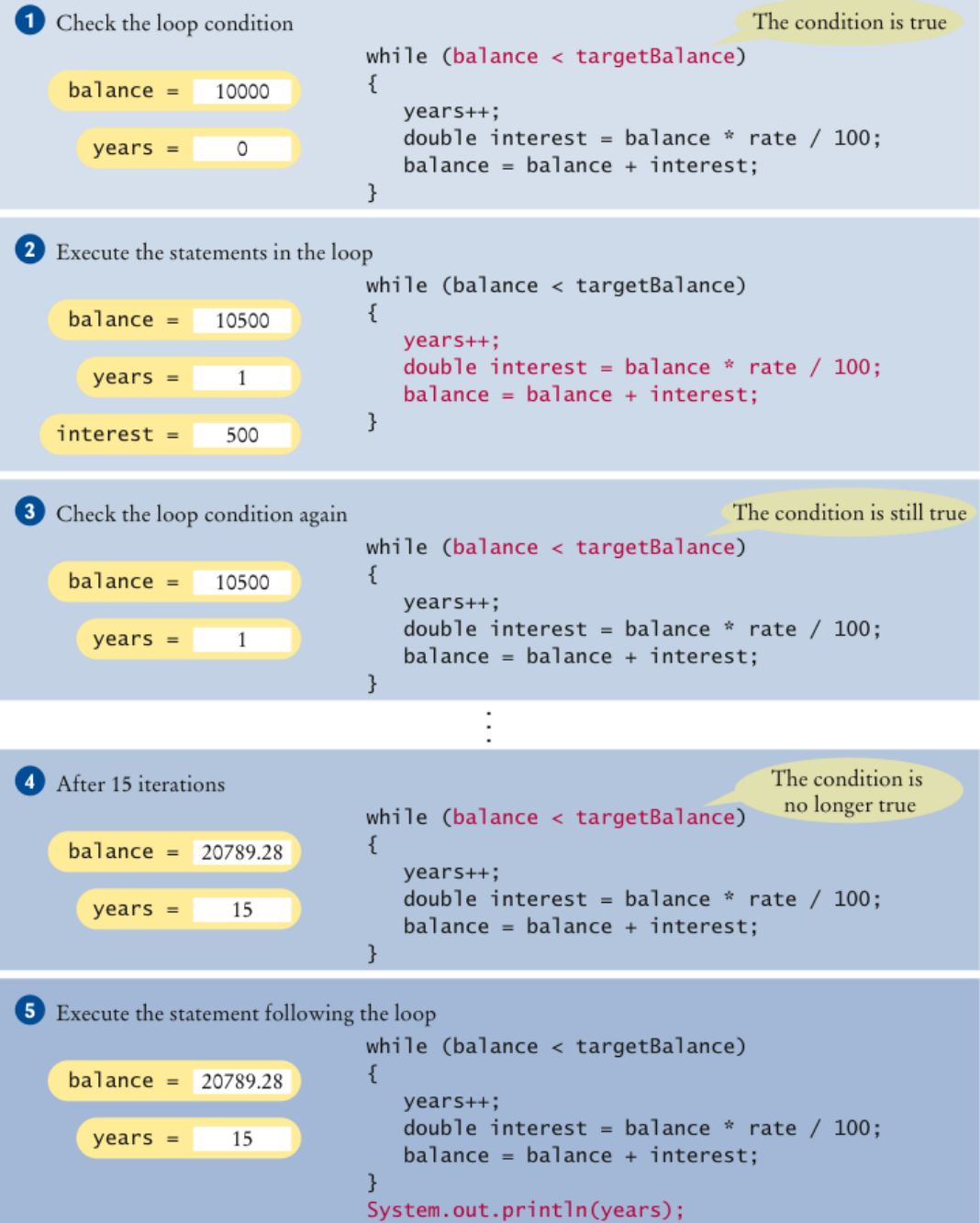
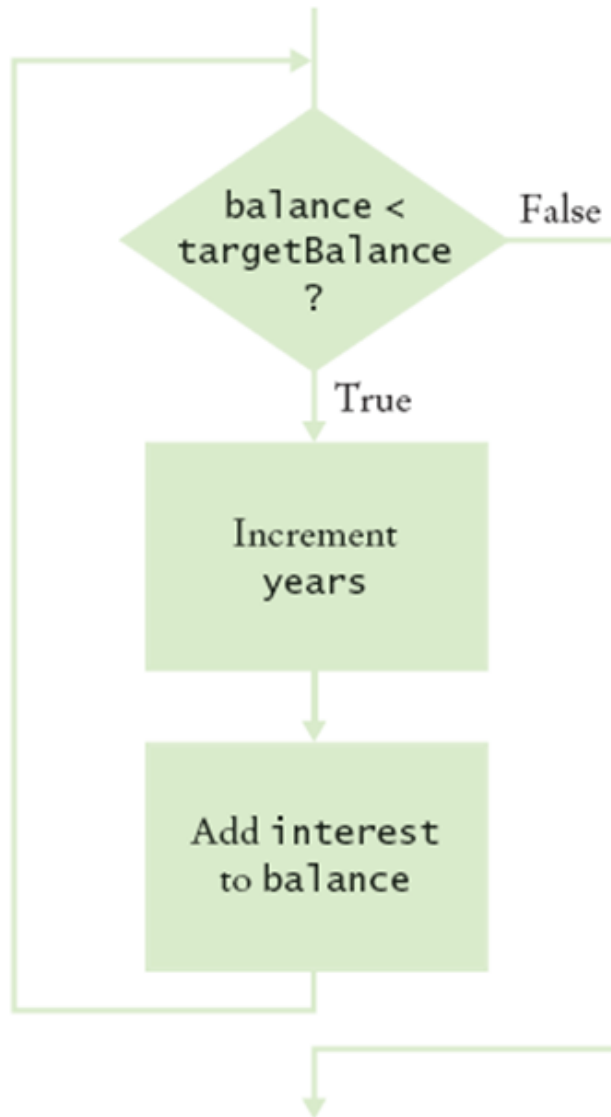


Figure 1 Execution of a while Loop

Diagramma di flusso per il ciclo while



File Investment.java

```
public class Investment {
    public Investment(double aBalance, double aRate) {
        balance = aBalance;
        rate = aRate;
        years = 0;
    }
    public double getBalance() {
        return balance;
    }
    public int getYears() {
        return years;
    }
    //accumula interessi fino a che il target è raggiunto

    public void waitForBalance(double targetBalance) {
        while (balance < targetBalance) {
            years++;
            double interest = balance * rate / 100;
            balance = balance + interest;
        }
    }
    private double balance;
    private double rate;
    private int years;
}
```


File InvestmentRunner.java

```
/**
   This program computes how long it takes for an investment
   to double.
 */
public class InvestmentRunner
{
    public static void main(String[] args)
    {
        final double INITIAL_BALANCE = 10000;
        final double RATE = 5;
        Investment invest = new Investment(INITIAL_BALANCE, RATE);
        invest.waitForBalance(2 * INITIAL_BALANCE);
        int years = invest.getYears();
        System.out.println("The investment doubled after " + years +
                           " years");
    }
}
```



Domande

- Quante volte viene eseguita l'istruzione nel seguente ciclo?

```
while (false) statement;
```

- Risposta: **Mai!**
- Cosa accade se la variabile **RATE** nel metodo **main** assume valore 0?

Risposta: **Il metodo `waitForBalance` va in loop**



Errori comuni: I loop infiniti

1. `int years = 0;`
 `while (years < 20) {`
 `balance = balance + balance * rate / 100;`
 `}`
2. `int years = 20;`
 `while (years > 0) {`
 `years++;`
 `}`



Errori comuni: scarto di 1

```
int years = 0;
while (balance < 2 * initialBalance)
{
    years++;
    double interest = balance * rate / 100;
    balance = balance + interest;
}
System.out.println("The investment reached the
    target after " + years + " years.");
```

- **years** dovrebbe iniziare da 0 o da 1?
- La condizione deve essere $< 0 \leq$



Usi comuni dei Loop

- Tenere traccia di un valore totale: una variabile a cui aggiungere ogni valore di input

```
double total = 0;
while (in.hasNextDouble())
{
    double input = in.nextDouble();
    total = total + input;
}
```



L'istruzione do/while

- Esegue il corpo del ciclo almeno una volta:

do

istruzione

while (*condition*);

- Esempio:

```
int value;
```

```
do
```

```
{
```

```
    String input = in.next();
```

```
    value = Integer.parseInt(input);
```

```
} while (value <= 0);
```



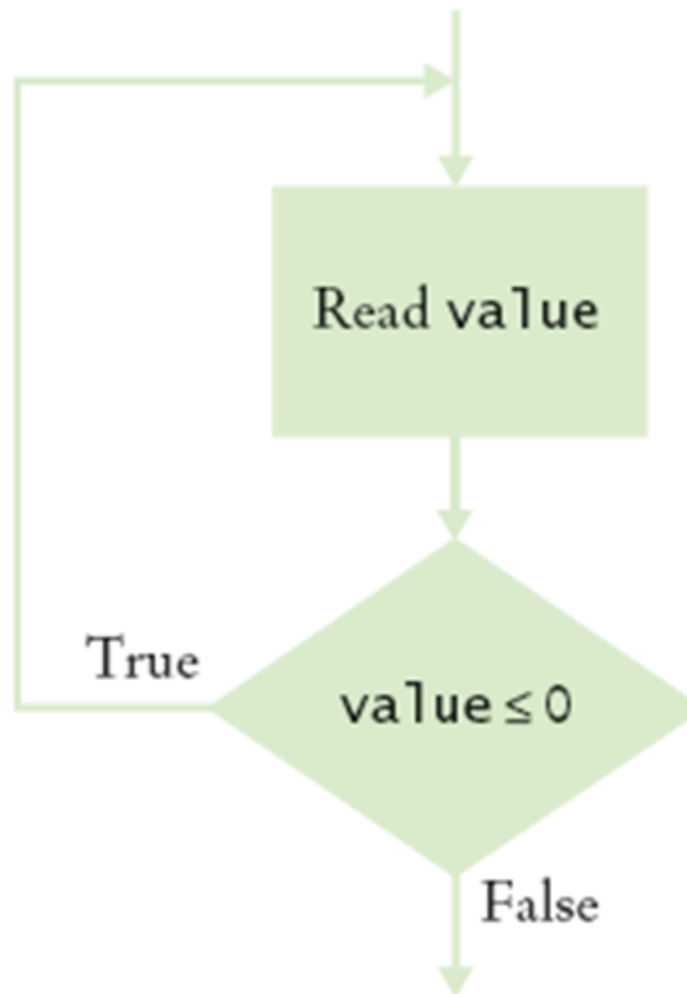
L'istruzione do/while

```
int value;  
do  
{  
    String input = in.next();  
    value = Integer.parseInt(input);  
} while (value <= 0);
```

Può essere riscritto con il seguente **while**:

```
boolean done = false;  
while (!done)  
{  
    System.out.print("Please enter a positive number: ");  
    value = in.nextDouble();  
    if (value > 0) done = true;  
}
```

Diagramma di flusso per do Loop





L'istruzione for

```
for (initialization; condition; update)  
    istruzione
```

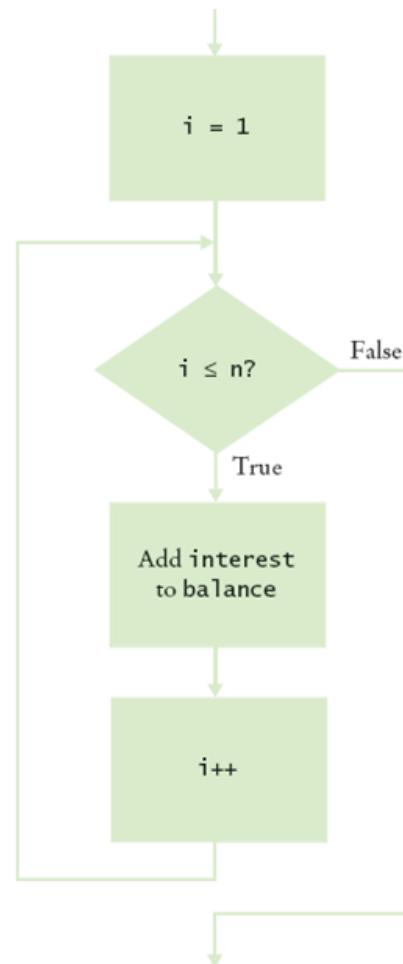
- Esempio:

```
for (int i = 1; i <= n; i++)  
{  
    double interest = balance * rate / 100;  
    balance = balance + interest;  
}
```

- Equivalente a

```
inizializzazione;  
while (condizione) {  
    istruzione; update;  
}
```

Diagramma di flusso ciclo for



```
for (years = n; years > 0; years--) . . .  
for (x = -10; x <= 10; x = x + 0.5) . . .
```



Esempio

- Aggiungiamo alla classe **Investment** il metodo **waitYears** che accumula gli interessi corrispondenti ad un certo numero di anni

```
public void waitYears(int n)
{
    for (int i = 1; i <= n; i++)
    {
        double interest = balance * rate / 100;
        balance = balance + interest;
    }
    years = years + n;
}
```



Errori comuni: punto e virgola!

- Un punto e virgola mancante

```
for (years = 1;  
    (balance = balance + balance * rate / 100) <  
    targetBalance;  
    years++)  
    System.out.println(years) ;
```

- Punto e virgola in più

```
sum = 0;  
for (i = 1; i <= 10; i++) ;  
    sum = sum + i;  
System.out.println(sum) ;
```



Loop annidati

- Esempio: stampiamo il triangolo

```
[]  
[] []  
[] [] []  
[] [] [] []
```

} n righe

```
for (int i = 1; i <= n; i++)  
{  
    // forma una riga del triangolo  
    for (int j = 1; j <= i; j++)  
        r = r + "[]";  
    r = r + "\n";  
}
```

Es.: lettura ciclica input (test interno)

```
import java.util.Scanner;
public class SommaInput{
    public static void main(String[] args) {
        double somma=0;
        Scanner in = new Scanner(System.in);
        System.out.println("Immetti valore oppure Q per uscire");
        boolean done = false;
        while (!done) {
            String input = in.next();
            if (input.equalsIgnoreCase("Q"))
                done = true;
            else {
                double x = Double.parseDouble(input);
                somma+=x;
            }
        }
        System.out.println("la somma e`:" + somma);
    }
}
```

Es.: lettura ciclica input (test inizio)

```
import java.util.Scanner;
public class SommaInput{
    public static void main(String[] args) {
        double somma=0;
        String input;
        Scanner in = new Scanner(System.in);
        System.out.println("Immetti valore oppure Q per uscire");
        while (!(input = in.next()).equalsIgnoreCase("Q")){
            double x = Double.parseDouble(input);
            somma+=x;
        }
        System.out.println("la somma e`:" + somma);
    }
}
```