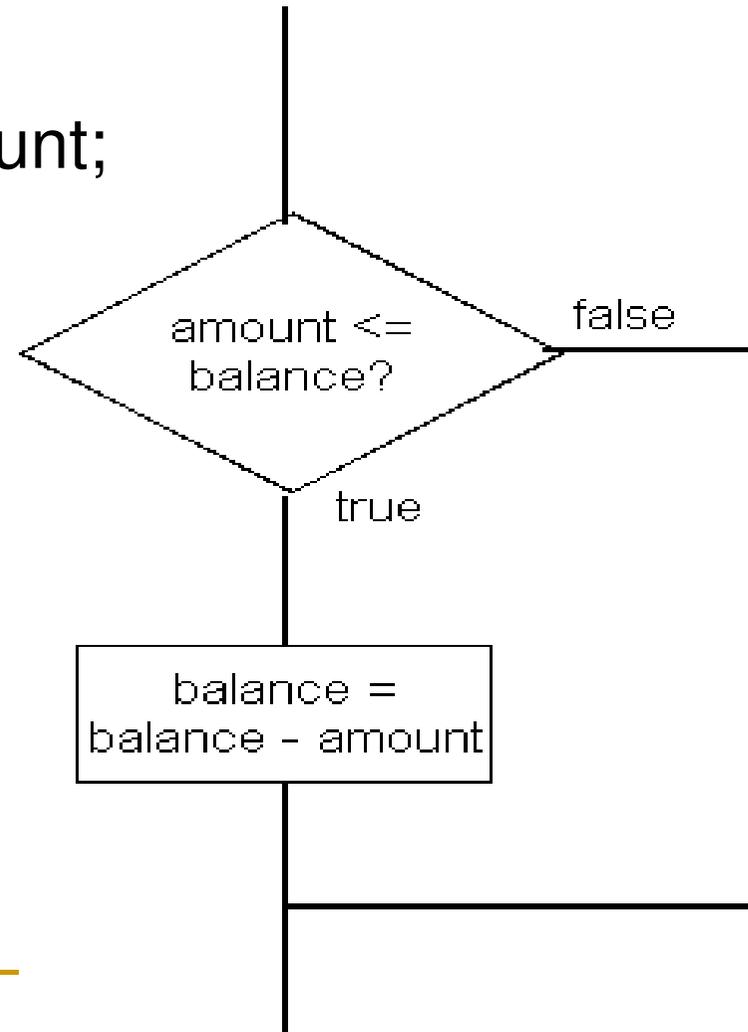

Decisioni

Istruzione if

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

Sintassi:

```
if (condizione)
    istruzione
```

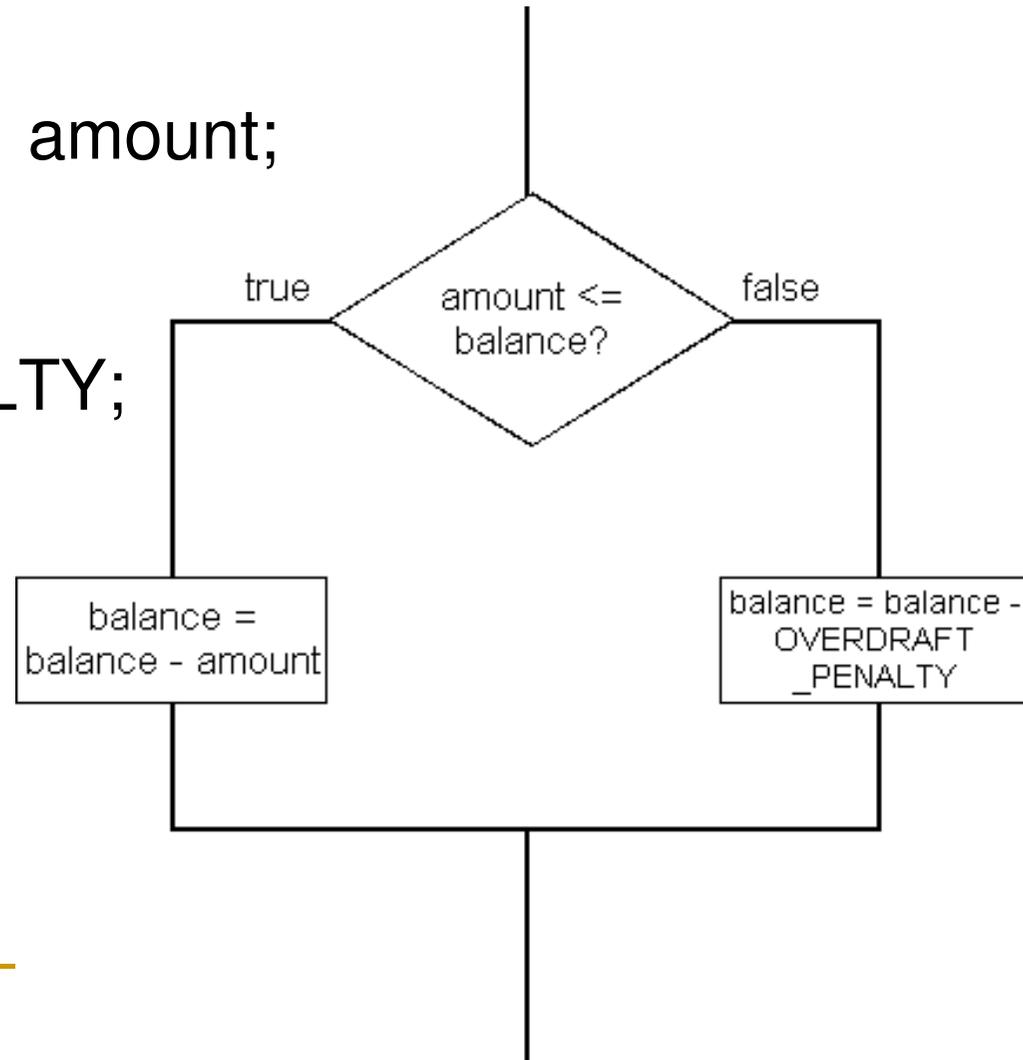


Istruzione if/else

```
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;
```

```
}
```

Confronto di numeri floating-point (1)

- Non è buona norma usare `==` per confrontare due numeri in virgola mobile
 - Arrotondamento dei valori
 - Espressioni matematiche con uguale valore potrebbero risultare diverse nella rappresentazione in virgola mobile
-

Confronto di numeri floating-point (2)

```
double r = Math.sqrt(2);
double d = r * r - 2;
if (d == 0)
    System.out.println("sqrt(2) al quadrato meno 2 è 0");
else
    System.out.println("sqrt(2) al quadrato meno
                        2 non è 0 ma " + d);
```

Output su console:

```
sqrt(2) al quadrato meno 2 non è 0 ma 4.440892098500626E-16
```

Confronto di numeri floating-point (3)

- Per verificare se due numeri floating point sono uguali si può verificare se la loro differenza in valore assoluto è minore di un valore soglia molto piccolo

$$|x - y| \leq \varepsilon$$

- ε è un valore prossimo a zero, come 10^{-14}
- Non va bene se x , y sono molto grandi o molto piccoli. In questo caso è meglio usare

$$|x - y| / \max(|x|, |y|) \leq \varepsilon$$

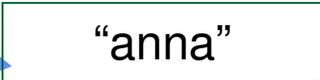
- Non si può usare se uno dei due numeri è 0
-

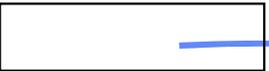
Confronto tra stringhe (1)

- Non bisogna usare `==` per confrontare due stringhe
if (stringa1 == stringa2)

// Testa se stringa1

//e stringa2 si riferiscono alla stessa stringa

stringa1=   

stringa2=   

stringa 1 == stringa 2
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
```

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";
```

```
.....
```

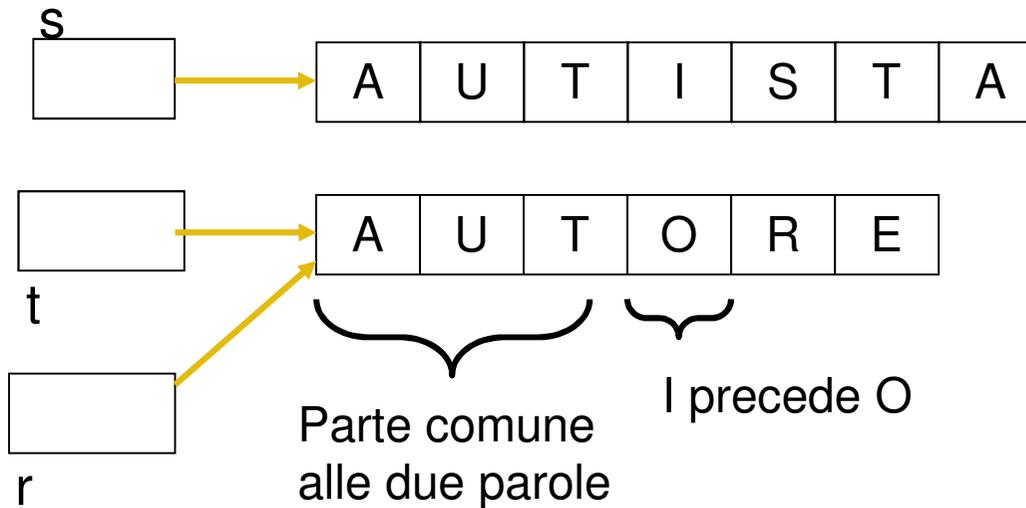
```
if (stringa1 == stringa2)
```

```
    //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

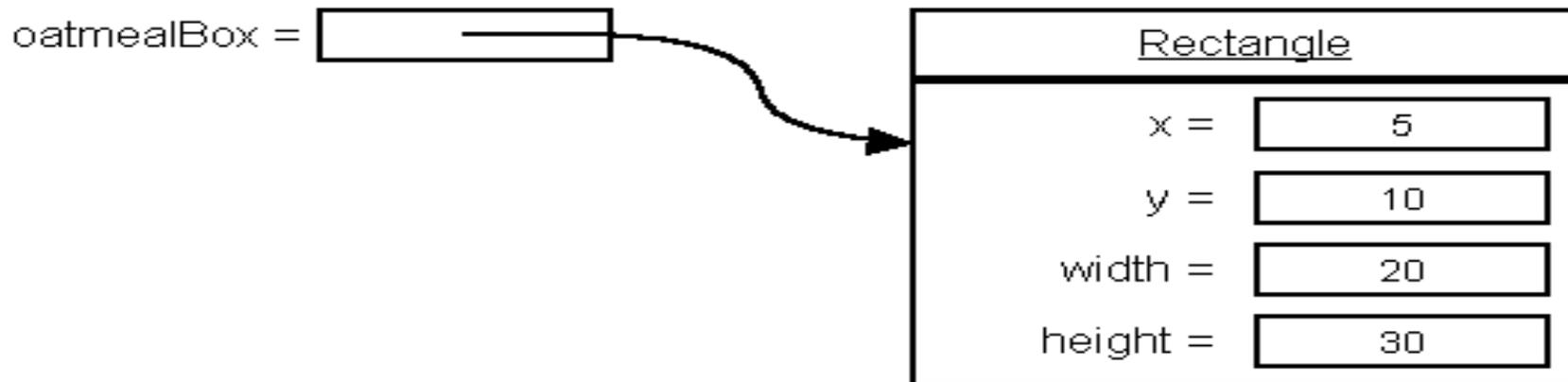
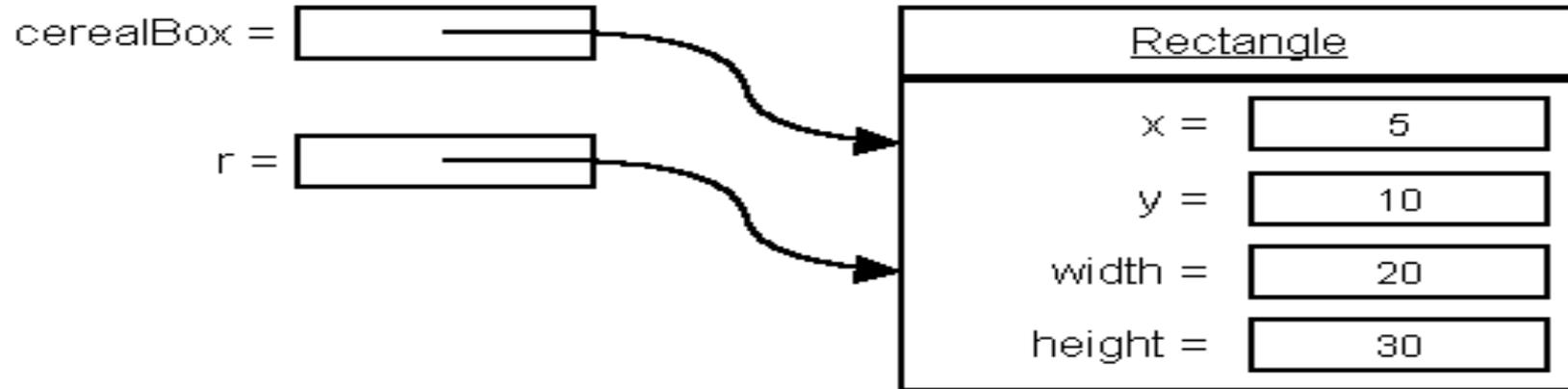
Confronto di oggetti (1)

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

```
Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
Rectangle oatmealBox = new Rectangle(5, 10, 20, 30);
Rectangle r = cerealBox;
if(cerealBox == oatmealBox)
    System.out.println("cerealBox e oatmealBox si
    riferiscono allo stesso rettangolo");
if(cerealBox.equals(oatmealBox))
    System.out.println("cerealBox e oatmealBox si
    riferiscono a rettangoli uguali");
if(r == cerealBox)
    System.out.println("r e cerealBox si riferiscono
    allo stesso rettangolo");
```

- Stampa: cerealBox e oatmealBox si riferiscono a rettangoli uguali
r e cerealBox si riferiscono allo stesso rettangolo

Confronto di oggetti (2)



Il metodo equals

- Quando si definisce una nuova classe bisogna definire un metodo **equals** 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 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) . . .`
-

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 *statement4*
 - Altra possibilità: **switch**
-

Switch

- A partire da Java 7, String si può usare come espressione nel costrutto switch (usa equals di String per il confronto)

```
switch (giorno.toLowerCase()) {  
    case "lun": dayNumber = 1; break;  
    case "mar": dayNumber = 2; break;  
    case "mer": dayNumber = 3; break;  
    case "gio": dayNumber = 4; break;  
    case "ven": dayNumber = 5; break;  
    case "sab": dayNumber = 6; break;  
    case "dom": dayNumber = 7; break;  
    default: dayNumber = 0; break;  
}
```

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
```

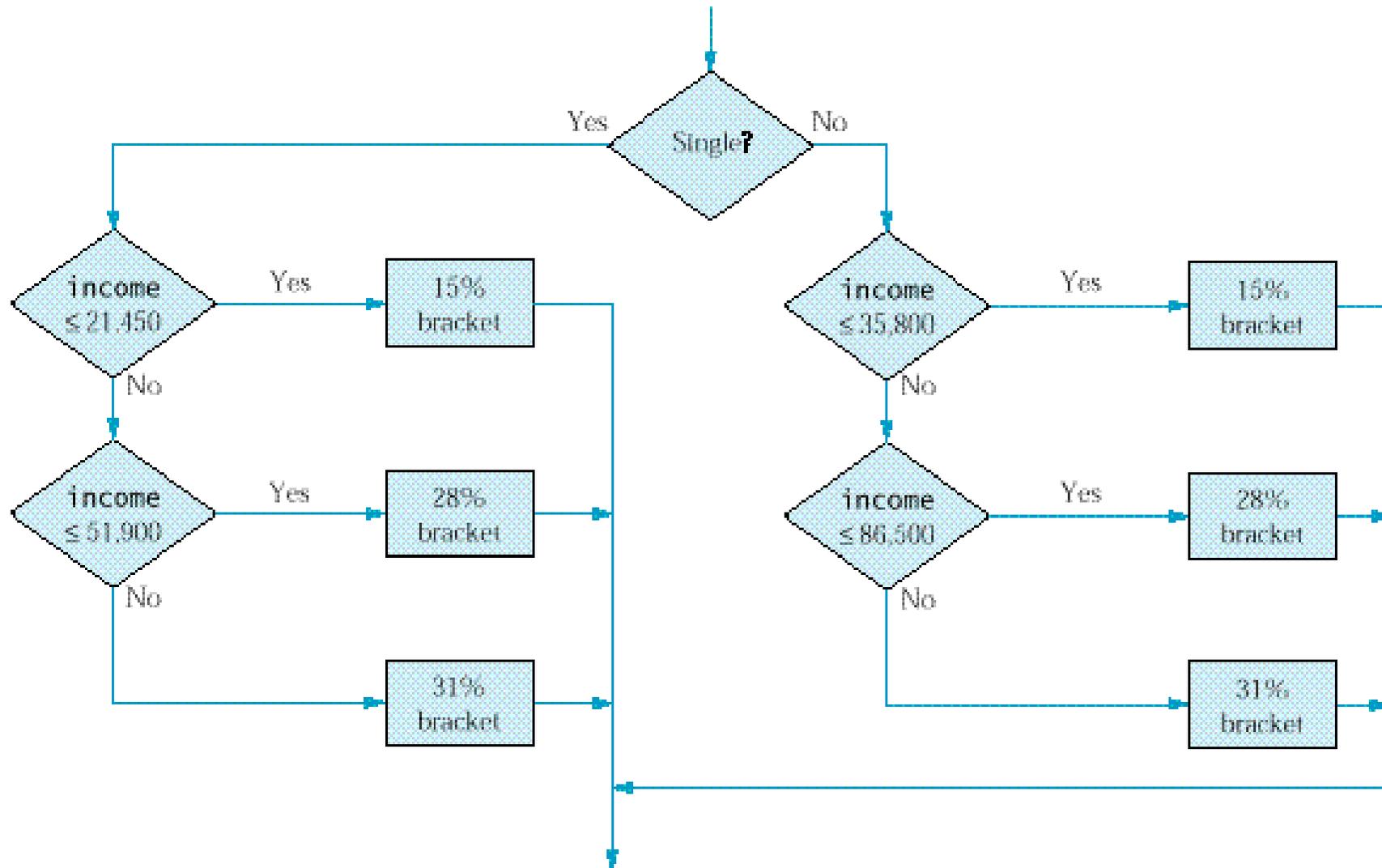
```
    private double richter;
```

```
}
```

Diramazioni annidate

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

Dichiarazione dei redditi



File TaxReturn.java

// Una dichiarazione dei redditi del 1992

```
public class TaxReturn
```

```
{
```

/* Costruisce un 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;
}
```

Ambiguità uso **else** (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
-

Ambiguità uso **else** (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**
-

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"
-

La legge di De Morgan

- La legge di De Morgan serve a semplificare le negazioni di espressioni contenenti *and* o *or*.
 - La legge di De Morgan stabilisce che
 - $!(A \ \&\& \ B)$ è uguale a $!A \ || \ !B$
 - $!(A \ || \ B)$ è uguale a $!A \ \&\& \ !B$
 - Si noti che $\&\&$ e $||$ vengono scambiati quando si portano le negazioni $!$ all'interno
 - Es.: $!(0 < \text{amount} \ \&\& \ \text{amount} < 1000)$ è uguale a $!(0 < \text{amount}) \ || \ !(\text{amount} < 1000)$, che è uguale a $0 \geq \text{amount} \ || \ \text{amount} \geq 1000$
-

Tipi enumerativi (1)

- Possiamo definire un tipo su un insieme finito di valori

```
public enum SettoreAuto {A, B, C};
```

- Definisce un tipo SettoreAuto che può assumere solo i valori A, B o C
 - Denotati SettoreAuto.A, SettoreAuto.B, etc.
 - La definizione deve avvenire in una classe
 - Per usarla fuori dalla classe

```
NomeClasse.SettoreAuto
```
 - Variabili di questo tipo si possono visualizzare nella console con il metodo `println()`
-

Tipi enumerativi (2)

- Per prendere in input i valori di un tipo enumerativo si invoca `valueOf` (già definito per questi tipi) per convertire una stringa al valore equivalente definito nel tipo enumerativo

`SettoreAuto.valueOf(input.next())`

- Se la stringa non appartiene ai valori definiti per il tipo, allora viene generata un'eccezione a run time

`IllegalArgumentException()`

- Conviene usare tipi enumerativi invece di interi se si vuole avere un controllo sui valori ammissibili
 - Ad esempio nel programma del calcolo tasse lo status poteva essere definito con un tipo enumerativo

File TaxReturn2.java

// Una dichiarazione dei redditi

```
public class TaxReturn2
```

```
{
```

/* Costruisce un dichiarazione dei redditi per un contribuente con
entrate pari al valore di anIncome e stato civile uguale a aStatus */

```
public TaxReturn(double anIncome, Status aStatus)
```

```
{
```

```
    income = anIncome;
```

```
    status = aStatus;
```

```
}
```

```
public double getTax()
{
    double tax = 0;
    if (status == 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 enum Status {SINGLE,MARRIED};
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 Status status;
}
```
