Espressioni booleane, confronti

Operatori relazionali

- All'interno delle parentesi tonde della condizione dell'**`if`** è possibile, come abbiamo visto, inserire il confronto tra due valori poiché questa espressione ha un valore di tipo boolean (**`true`** o **`false`**)
- Vediamo tutti gli operatori relazionali che possiamo usare nella seguente tabella

Operatori Relazionali

Operatore Java	Notazione matematica	Descrizione
>	>	Maggiore
>=	≥	Maggiore o uguale
<	<	Minore
<=	≤	Minore o uguale
==	=	Uguale
!=	≠	Diverso

Errore comune: = invece di ==

- I programmatori ancora inesperti spesso confondono l'uso di **=** con **==**
- In particolare un errore comune è quello di inserire in un confronto l'operatore **=** invece che **==**
- **`if (a=5) b++;`** // Errore di compilazione
- L'espressione **`a=5`**, **oltre ad essere un assegnamento e non un confronto**, ha un valore di tipo **`int`**

Confronti di valori in virgola mobile

- Gli operatori **==** o **`!=`** non hanno molto senso applicati a valori in virgola mobile
- Un errore tipico è quello di cercare di controllare se un valore **`double`** (o **`float`**) sia uguale a **`0.0`**
- È molto improbabile che un **`double`** risultante da un calcolo sia esattamente zero
- Potrebbe essere invece un valore prossimo allo zero

Confronti di valori in virgola mobile

- Prendiamo ad esempio questo semplice programma:

```
double r = Math.sqrt(2);
double d = r * r -2;
if (d == 0)
  System.out.println("Radice quadrata di 2 per 2 meno 2 fa 0");
else
  System.out.println("Radice quadrata di 2 per 2 meno 2 fa " +
                                                        d);
```

- Il valore stampato non è zero: **`d`** vale **`4.440892098500626E-16`** che è prossimo a zero, ma non è zero!

Confronti di valori in virgola mobile

- In generale, per confrontare l'uguaglianza di due valori in virgola mobile è bene fissare una soglia EPSILON di tolleranza e vedere se i due valori sono sufficientemente prossimi rispetto a questa
- Per esempio si può definire

```
final double EPSILON = 1E-14;
```

- E poi controllare se |x-y| ≤ EPSILON. Se è vero si può decidere che i due valori vanno considerati "uguali"

Confronti di valori in virgola mobile

- Tuttavia, se x e y sono valori molto grandi, la loro differenza potrebbe essere una quantità maggiore di EPSILON, ma comunque, vista la loro grandezza, essere irrisoria
- È bene, quando x e y non sono prossimi a zero, considerare la differenza dei due valori rapportata alla loro gandezza:

$$\frac{|x-y|}{\max(|x|,|y|)} \leq \varepsilon$$

Confronti di valori in virgola mobile

- In codice java:

```
final double EPSILON = 1E-14;
double x, y;
...
if (Math.abs(x-y) /
  Math.max(Math.abs(x), Math.abs(y)) <=
  EPSILON)
    System.out.println("\"Uguali\"");
else System.out.println("\"Diversi\"");
```


Confronto di Stringhe

- Le stringhe in Java, lo sappiamo, sono oggetti
- Pertanto il valore di una variabile di frame di tipo **`String`** non è la stringa in sé, ma un riferimento all'oggetto stringa
- Se cercassimo di confrontare direttamente due variabili di tipo stringa staremmo semplicemente controllando se puntano allo stesso oggetto
- In quel caso sono sicuramente uguali, come stringhe, ma in genere ciò che ci interessa è sapere se il contenuto di due stringhe (oggetti stringa diversi) è lo stesso

Confronto di Stringhe

```
...
String pippo = console.readLine();
String pluto = console.readline();
if (pippo == pluto) //falso
  System.out.println("Stringhe Uguali")
else System.out,println("Stringhe Diverse")
```

- Per il confronto del contenuto di due stringhe dobbiamo usare il metodo **`equals`** della classe **`String`**
- Come tipico della programmazione ad oggetti, un operatore binario fra due oggetti viene realizzato con un metodo che va chiamato su uno dei due e a cui va passato un riferimento all'altro oggetto operando

Confronto di Stringhe

```
System.out.println("Inserisci una stringa");
String pippo = console.readLine();
System.out.println("Inserisci una seconda stringa per il confronto");
String pluto = console.readLine();
if ( pippo == pluto )
  System.out.println("Confronto con == : Stringhe Uguali");
else
  System.out.println("Confronto con == : Stringhe Diverse");
if ( pippo.equals(pluto) )
  System.out.println("Confronto con metodo equals: Stringhe" + "
   Uguali");
else
  System.out.println("Confronto con metodo equals: Stringhe" + "
   Diverse");
```

- Inserendo due stringhe uguali il primo confronto fallisce, mentre il secondo ha successo

Confronto di Stringhe

- La classe **`String`** fornisce anche operatori di confronto “corrispondenti” a <, >
- Il metodo **`compareTo`** ha un parametro **`String`** in ingresso e confronta la stringa su cui è chiamato con la stringa passata come parametro
- Il valore di uscita è un **`int`** il cui valore indica se la stringa passata è uguale, “maggiore” o “minore” nel senso di ordine **alfabetico**

Confronto di Stringhe

```
int r = string1.compareTo(string2);
```

- Se **`r > 0`** allora **`string1`** precede **`string2`** nell’ordine alfabetico
- Se **`r < 0`** allora **`string1`** segue **`string2`** nell’ordine alfabetico
- Se **`r == 0`** allora le stringhe sono uguali (alternativa a **`equals`**)

Confronto di oggetti

- Il discorso fatto per le stringhe si applica agli oggetti in generale
- Applicando l’operatore **==** a due variabili riferimento si controlla semplicemente se puntano allo stesso oggetto
- Se si vuole invece confrontare lo stato si deve fornire la classe di un metodo **`equals`** simile a quello che viene fornito con **`String`**