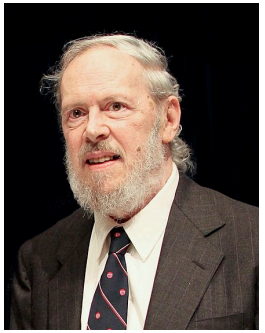
# Primo programma in Java

# Il nostro primo programma Java

- Tradizionalmente, in onore di Dennis Ritchie, inventore del linguaggio C, il primo programma che si scrive quando si impara un linguaggio di programmazione ha il compito di visualizzare sullo schermo un semplice saluto al mondo

```
Hello, World!
```



- Scriviamo il programma nel file **Hello.java**

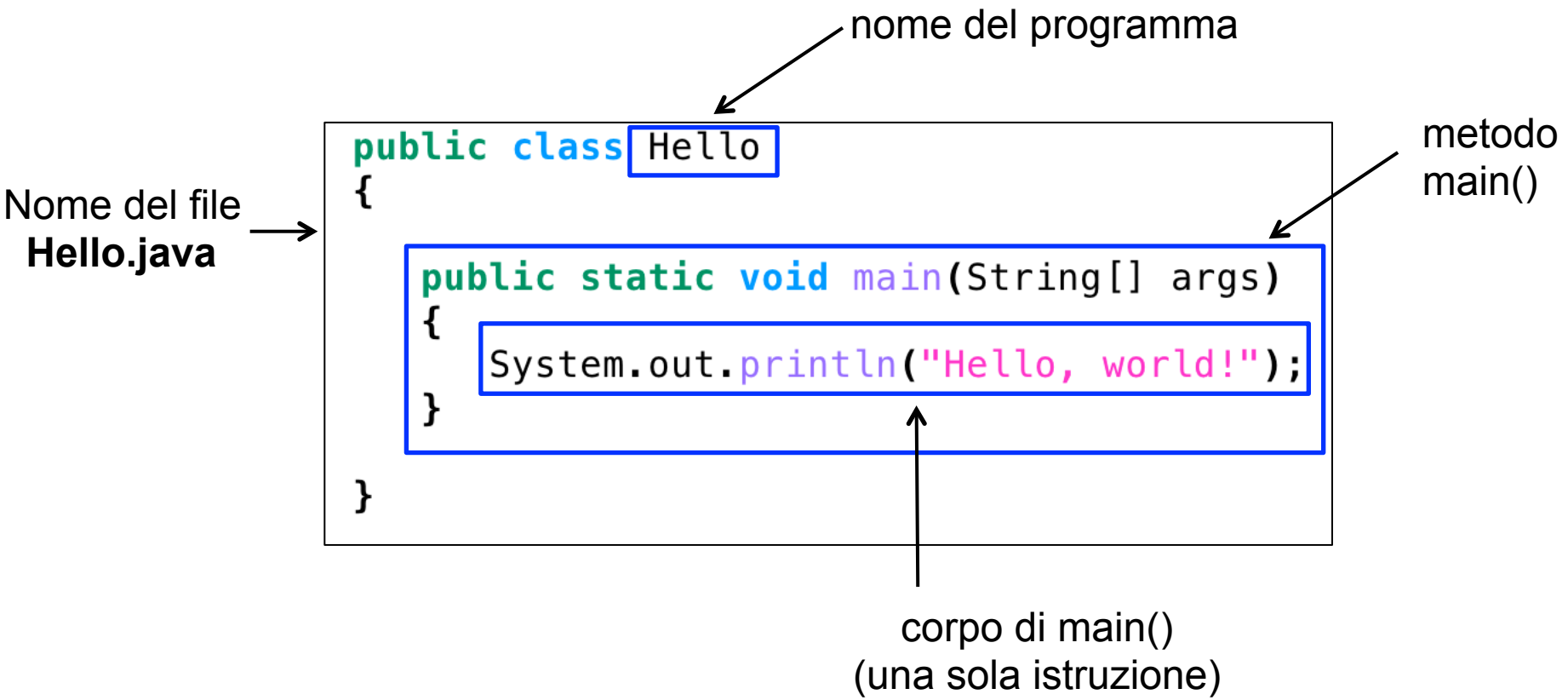
usando un editor di testi (che NON introduca segnali di controllo per la formattazione, cioè NON Word, ma, ad esempio, WordPad in Windows, gedit in Linux, …)

# Hello.java

```
public class Hello
{

   public static void main(String[] args)
   {
      System.out.println("Hello, world!");
   }

}
```


nome del programma
Nome del file
Hello.java
metodo
main()
public class Hello
{
public static void main(String[] args)
{
System.out.println("Hello, world!");
}
}
corpo di main()
(una sola istruzione)

# Il nostro primo programma Java

```
public class Hello
{  public static void main(String[] args)
   {
     System.out.println("Hello, World!");
   }
}
```

- Occorre fare molta attenzione
  - il testo va inserito esattamente come è presentato
    - per il momento…
  - ***maiuscole e minuscole sono considerate distinte***
  - il file *deve* chiamarsi **Hello.java**

# Il nostro primo programma Java

- A questo punto, in una "finestra di comandi" (*shell*) ***compiliamo*** il programma

```
javac Hello.java
```

  e il compilatore genera il file **Hello.class**

- Ora ***eseguiamo*** il programma

```
java Hello
```

  ottenendo la visualizzazione del messaggio di saluto sullo schermo

```
Hello, World!
```

```
public class Hello
{  public static void main(String[] args)
   {  System.out.println("Hello, World!");
   }
}
```

# Analisi del primo programma

- La prima riga `public class Hello`

  ***definisce*** una nuova ***classe***

- Le classi sono **contenitori o *fabbriche di oggetti*** e rappresentano un concetto fondamentale in Java, che è un linguaggio di programmazione ***orientato agli oggetti*** (**OOP**, ***O**bject-**O**riented **P**rogramming*)
  - Per il momento, consideriamo gli ***oggetti*** come ***elementi da manipolare in un programma Java***
  - Classi correlate possono essere raggruppate per formare un ***package***

# Analisi del primo programma

- La ***parola chiave*** **public** indica che la classe **Hello** può essere utilizzata da tutte le altre classi (in seguito vedremo **private**…)

```
public class Hello
```

- **Una parola chiave è una parola "riservata" del linguaggio che va scritta esattamente così com'è e che non può essere usata per altri scopi**
- La parola chiave **class** indica che inizia la **definizione** di una ***classe***
  - **IMPORTANTE**: Ciascun ***file sorgente*** (che faccia parte di un programma Java) può contenere ***una sola classe pubblica***, il cui nome ***deve coincidere*** con il nome del file

# Analisi del primo programma

- **Per ora** non usiamo le classi come fabbriche di oggetti, ma come ***contenitori di metodi***
- Un **metodo** serve a definire una sequenza di istruzioni o ***enunciati*** che ***descrive come svolgere un determinato compito*** (in altri linguaggi i metodi si chiamano *funzioni* o *procedure*)
  - In pratica, definisce un algoritmo o una sua parte!
- In Java, un metodo ***deve*** (sintatticamente) essere inserito in una classe, quindi
  - le classi rappresentano il meccanismo principale per l'organizzazione dei programmi in Java

```
public class Hello
{  public static void main(String[] args)
   {  System.out.println("Hello, World!");
   }
}
```

# Metodo main

- La costruzione

```
public static void main(String[] args)
{   ...
}
```

definisce il metodo **main** (cioè *principale*)

- Un programma Java ***deve*** avere un metodo **main**
- Anche qui, **public** significa ***utilizzabile da tutti***
- Invece, **static** significa che il metodo **main** ***non esamina e non modifica gli oggetti della classe*** **Hello** a cui appartiene

# Sintassi di un semplice programma

```
public class NomeClasse
{  public static void main(String[] args)
   {  enunciati
   }
}
```

- Scopo: eseguire un programma semplice, descritto da ***enunciati*** e contenuto nel file ***NomeClasse.java***
  - Semantica: gli enunciati che costituiscono un metodo vengono eseguiti nell'ordine in cui sono scritti
- Nota: la parte in **blu** viene per ora considerata una ***infrastruttura necessaria***, approfondita in seguito

# Analisi del primo programma

- Il **corpo** di un metodo è racchiuso tra parentesi graffe
- Gli enunciati del ***corpo*** di un metodo vengono eseguiti uno alla volta *nella sequenza in cui sono scritti*
- Ogni enunciato termina con il carattere `;`

```
System.out.println("Hello, World!");
```

- Il metodo **main** del nostro esempio ha un solo enunciato, che visualizza una riga di testo
- Ma *dove* la visualizza? Un programma può inserire testo in una finestra, scriverlo in un file o anche inviarlo ad un altro computer attraverso Internet...

# Analisi del primo programma

```
System.out.println("Hello, World!");
```

- Nel nostro caso la destinazione è l'***output_standard*** (o "flusso di uscita standard")
  - è una proprietà di ciascun programma che dipende dal sistema operativo del computer
- All'interno di un programma Java, l'output standard è rappresentato da un ***oggetto*** di nome **out**
  - come ogni metodo, ***anche gli oggetti devono essere inseriti in classi***: **out** è inserito nella classe **System** della **libreria standard**, che contiene oggetti e metodi da utilizzare per accedere alle ***risorse di sistema***
  - per ***usare*** l'oggetto **out** della classe **System** si scrive

```
System.out
```

# Analisi del primo programma

```
System.out.println("Hello, World!");
```

- Quando si ***usa*** un oggetto, bisogna specificare ***cosa*** si vuol fare con l'oggetto stesso
  - in questo caso vogliamo ***usare un metodo*** dell'oggetto **out**, il metodo **println**, che stampa una riga di testo
  - per ***usare*** il metodo **println** dell'oggetto **System.out** si scrive `System.out.println(parametri)`
  - la coppia di parentesi tonde racchiude le informazioni necessarie per l'esecuzione del metodo (***parametri***)

# Attenzione

```
System.out.println("Hello, World!");
```

- A volte il carattere ***punto*** significa "usa un oggetto di una classe", altre volte "usa un metodo di un oggetto": dipende dal contesto...
  - Molte grammatiche (formali o naturali) hanno una sintassi dipendente dal contesto

# Invocazione di metodo (non statico)

- Sintassi:

```
oggetto.nomeMetodo(parametri)
```

- Scopo: invocare il metodo ***nomeMetodo*** dell'***oggetto***, fornendo ***parametri*** se sono richiesti
- Nota: se **non** sono richiesti parametri, le parentesi tonde devono essere indicate ugualmente
- Nota: se ci sono due o più parametri, essi vanno separati l'uno dall'altro con una ***virgola***, all'interno delle parentesi tonde
  - oggetto.nomeMetodo(arg1,arg2,arg3);

# Analisi del primo programma

- Una sequenza di caratteri racchiusa tra virgolette si chiama stringa (*string*)

```
"Hello, World!"
```

- C'è un motivo per dover racchiudere le stringhe tra virgolette
  - come farebbe altrimenti il compilatore a capire che volete, ad esempio, visualizzare la parola **class** e non che volete iniziare la definizione di una nuova classe?
- Il metodo **println** può anche stampare numeri o **risultati** di operazioni aritmetiche

```
System.out.println(7);
```

```
System.out.println(3 + 4);
```

- C'è anche il metodo **print**, che funziona come **println** ma ***non va a capo al termine della stampa***

# Esempio

Perché questa sequenza di enunciati diventi un programma bisogna ovviamente inserirla nella "solita" infrastruttura

- Cosa visualizza questo enunciato?

```
System.out.println(3 + 4);
```

- E questo?

```
System.out.println("3 + 4");
```

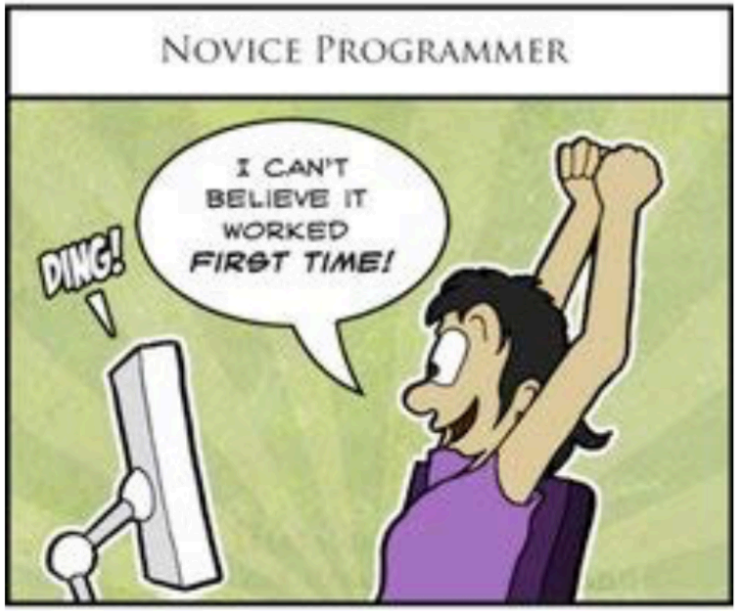
- Possono servire entrambi?

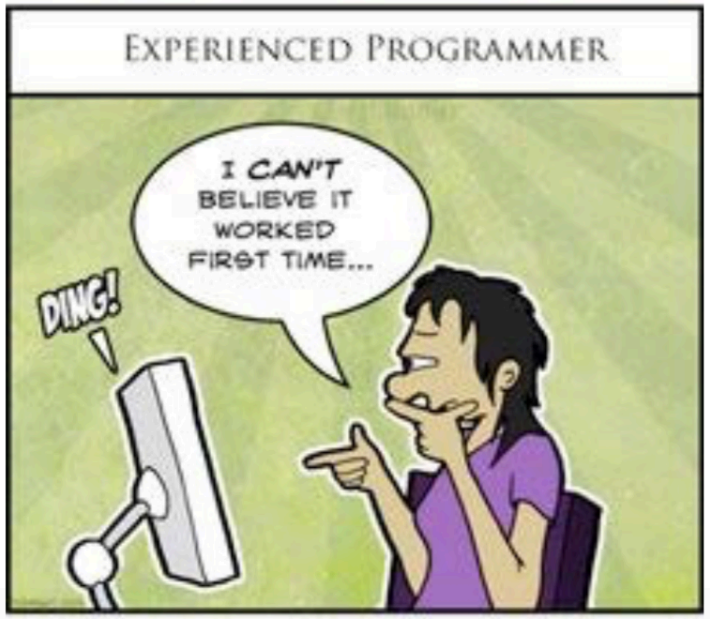
```
System.out.print("L'operazione ");
System.out.print("3 + 4");
System.out.print(" ha come risultato: ");
System.out.println(3 + 4);
```

```
L'operazione 3 + 4 ha come risultato: 7
```

Attenzione agli spazi…

# Errori di programmazione



# Errori di programmazione

```
System.out.println("Hello, World!");
```

- L'attività di programmazione, come ogni altra ***attività di progettazione***, è soggetta a errori di vario tipo

  - **errori di sintassi o di compilazione**

```
System.aut.println("Hello, World!");
```

```
System.out.println("Hello, World!);
```

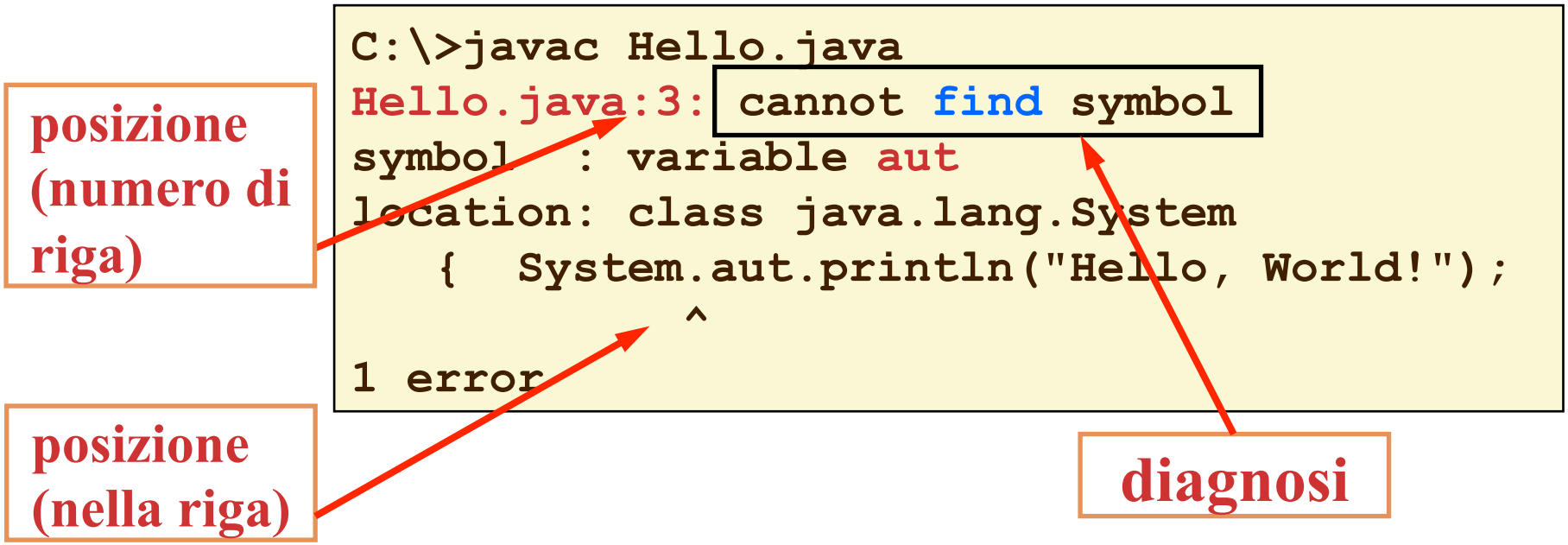
  - **errori logici o di esecuzione**

```
System.out.println("Hell, World!");
```

# Errori di sintassi

```
System.aut.println("Hello, World!");
```

- In questo caso il compilatore riesce agevolmente ad individuare e segnalare l'errore di sintassi, perché identifica il nome di un oggetto (***simbolo***) che non è stato definito (**aut**) e che, quindi, non riesce a **trovare**



# Errori di sintassi

```
System.out.println("Hello, World!);
```

virgolette mancanti

- Questo è invece un caso molto più complesso: viene giustamente segnalato il **primo errore**, una stringa non terminata, e viene evidenziato il punto dove ***inizia*** la stringa

```
C:\>javac Hello.java
Hello.java:3: unclosed string literal
    {  System.out.println("Hello, World!);
                          ^
Hello.java:4: ')' expected
    }
    ^
2 errors
```

# Errori di sintassi

- Viene però segnalato anche un **secondo errore**
  - il compilatore ***si aspetta di trovare una parentesi tonda chiusa***, in corrispondenza di quella aperta
  - ***la parentesi in realtà c'è***, ma il compilatore l'ha inserita all'interno della stringa, cioè ***ha immaginato di prolungare la stringa fino al termine della riga***

```
C:\>javac Hello.java
Hello.java:3: unclosed string literal
    {  System.out.println("Hello, World!);
                                ^
Hello.java:4: ')' expected
    }
    ^
2 errors
```

# Errori multipli

- Quando il compilatore segnala più errori, c'è la possibilità che le segnalazioni successive alla prima siano sbagliate
  - Il compilatore viene "ingannato" dal primo errore
- Finché non si ha acquisito una certa esperienza, **bisogna correggere soltanto il primo degli errori segnalati dal compilatore**
  - Poi si salva il file modificato, si compila nuovamente e si osservano i nuovi, eventuali errori
  - **Nel caso precedente, correggendo il primo errore segnalato dal compilatore, il secondo "sparisce"**

# Errori logici

```
System.out.println("Hell, World!");
```

manca un carattere

- Questo errore, invece, ***non viene segnalato dal compilatore***, che non può sapere che cosa il programmatore abbia intenzione di far scrivere al programma sull'output standard
  - ***la compilazione va a buon fine***
  - si ha un errore durante l'***esecuzione*** del programma, perché viene prodotto un output **diverso** dal previsto

```
C:\>java Hello
Hell, World!
```

# Errori logici

- Sono molto più insidiosi degli errori di sintassi
  - ***il programma*** viene compilato correttamente,
    ma ***non fa quello che dovrebbe fare***
- L'eliminazione degli errori logici richiede molta ***pazienza***, eseguendo il programma e osservando con attenzione i risultati prodotti
  - ***è necessario collaudare i programmi, come qualsiasi altro prodotto dell'ingegneria***
- Si usano programmi specifici (***debugger)*** per trovare gli errori logici (***bug)*** in un programma
  - **noi non useremo un debugger**

# Modifica-Compila-Collauda