# Laboratorio di Programmazione II

## Corso di Laurea in Bioinformatica

## Dipartimento di Informatica - Università di Verona

**Autore: Alessandro Farinelli**
**Corso di Laboratorio di Programmazione II**
**http://profs.sci.univr.it/~farinelli/pubs/publications.html**

# Sommario





- Presentazione del corso
- Concetti base linguaggio Java
  - Editare, compilare e correggere programmi Java
  - Oggetti e Metodi
  - Variabili

# Presentazione del Corso





**Autore: Alessandro Farinelli**
**Corso di Laboratorio di Programmazione II**
**http://profs.sci.univr.it/~farinelli/pubs/publications.html**

# Docente

## Contatti


- Alessandro Farinelli
- Dipartimento di Informatica
  Ca Vignal 1 - Primo piano - Stanza 1.55
  Tel. 0458027072 e-mail alessandro.farinelli@univr.it

- Ricevimento:
  - Mercoledi' 16:00 – 18:00
  - Su appuntamento tramite e-mail

# Corso



## Dati del Corso

- Secondo modulo del corso di Algoritmi
- 64 ore di laboratorio
- 22 lezioni, 3 ore per lezione
  - Martedi' 08:30 – 11:30
  - Venerdi' 14:30 – 17:30

# Esame



## Modalità esame

- L'esame vale il 50% del voto dell'intero corso di Algoritmi
- Due modalità di esame:
  - Prove parziali
  - Esame singolo
- Prove parziali:
  - Valido solo per il primo appello che segue la fine delle lezioni (I appello di Febbraio)
  - Prova al calcolatore (probabilmente fine Novembre)
  - Progetto (svolgimento durante la lezione, presentazione orale a fine corso)
  - voto finale = 50% progetto + 50% prova calcolatore
- Esame singolo
  - Vale per qualsiasi sessione
  - prova singola al calcolatore di difficoltà pari alla somma delle difficoltà delle prove parziali.

# Pre-requisiti



## Conoscenze di base

- Conoscenza di base del linguaggio Java
- Capacità di editare compilare eseguire un sorgente java
- Conoscenza degli argomenti sviluppati nel modulo di teoria

# Programma



## Programma del corso

- Basi Java
- Ricorsione
- Ordinamento
- Strutture dati ed algoritmi notevoli:
  - Strutture lineari (Pile, code, liste)
  - Alberi
  - Grafi
- Algoritmi notevoli per la bioinformatica (PDP, ricerca motivi regolatori, sottosequenza comune)
- Basi Matlab per analisi e visualizzazione dati

# Materiale Didattico



## Materiale didattico di riferimento

- Slide del corso
  - Disponibili sul sito del corso
- Libri consigliati
- Manuali Java esempio:
  - Thinking in Java, Bruce Eckel
  - Java API

# Come reperire i file durante la lezione



## Accesso ai file

- Sito del corso
- codice
- Copiare i file in locale per modificarli

# Il linguaggio Java





**Autore: Alessandro Farinelli**

# Paradigmi di programmazione



## Principali paradigmi di programmazione

- si distinguono per l'enfasi che pongono sui due aspetti fondamentali: oggetti e operazioni.
  1. **imperativo**: enfasi sulle operazioni intese come azioni/comandi/istruzioni; esempio: C, Pascal
  2. **funzionale**: enfasi sulle operazioni intese come funzioni che calcolano risultati; esempio: Lisp, Prolog
  3. **orientato agli oggetti**: enfasi sugli oggetti che complessivamente rappresentano il dominio di interesse; Java, C++ (quasi...)
- Maggior parte dei programmi mettono a disposizione strutture per realizzare diversi paradigmi

# Il linguaggio Java



## Linguaggio Java

- alto livello
  - costrutti del linguaggio risparmiano molta fatica al programmatore
- orientato agli oggetti: tutto è un oggetto!
  - supporta i paradigmi imperativo e funzionale
- indipendente dalla piattaforma
  - grazie ai concetti di bytecode e Virtual Machine

# Un semplice programma Java



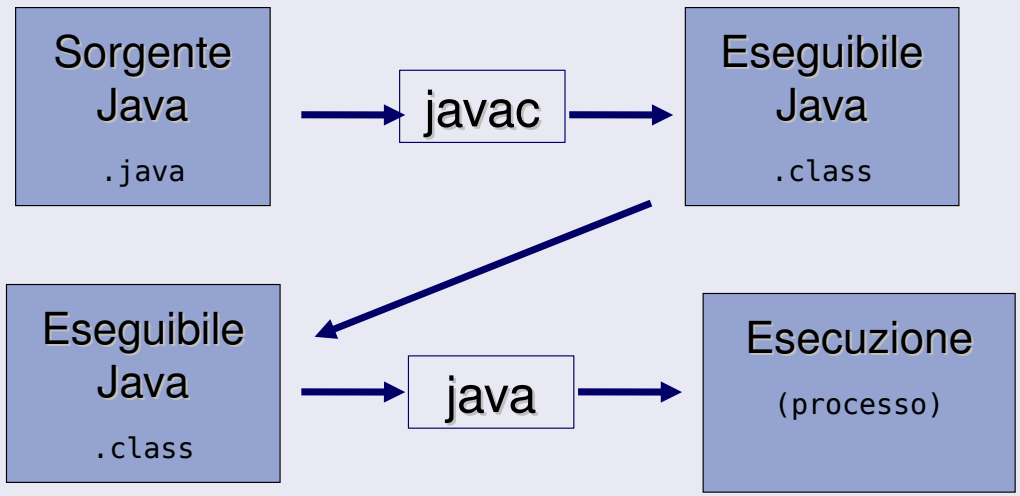
## programma java

```
import java.lang.*;
public class Semplice {
  public static void main(String[] args) {
    System.out.println("Questo è un semplice programma Java.");
  }
}
```

# Compilare ed eseguire codice Java



## Compilare ed eseguire un programma in Java



# Scrivere compilare ed eseguire codice Java



## Dall'editor all'esecuzione

1. Scrittura
   - Scrivere il programma con un qualsiasi editor, e.g, notepad (windows), gedit (linux), etc.
   - Salvare il file con **NomeClasse.java**
   - Il nome del file deve essere uguale al nome della classe

# Scrivere compilare ed eseguire codice Java



## Dall'editor all'esecuzione

2 Compilazione

- Chiamare il compilatore java con parametro il nome del file da compilare
- Usando il Java Standard Developmnent Kit il compilatore viene chiamato con il comando **javac**
- **javac NomeClasse.java** produce **NomeClasse.class**
- **NomeClasse.class** rappresenta il bytecode del programma java

# Scrivere compilare ed eseguire codice Java



## Dall'editor all'esecuzione

3 Esecuzione

- Chiamare il programma java che interpreta il bytecode con parametro il nome del file compilato senza .class
- Usando il Java Standard Developmnent Kit il comando per eseguire un programma java compilato è **java**
- **java NomeClasse** comporta l'esecuzione di NomeClasse

# Errori



## Un programma errato

```
public class Errori {
  public static void main(String[] args) {
    System.out.println("Questi sono i miei primi errori Java")
    Sistem.out.println("...e non saranno gli ultm!!!");
  }
}
```

# Tipi di Errori



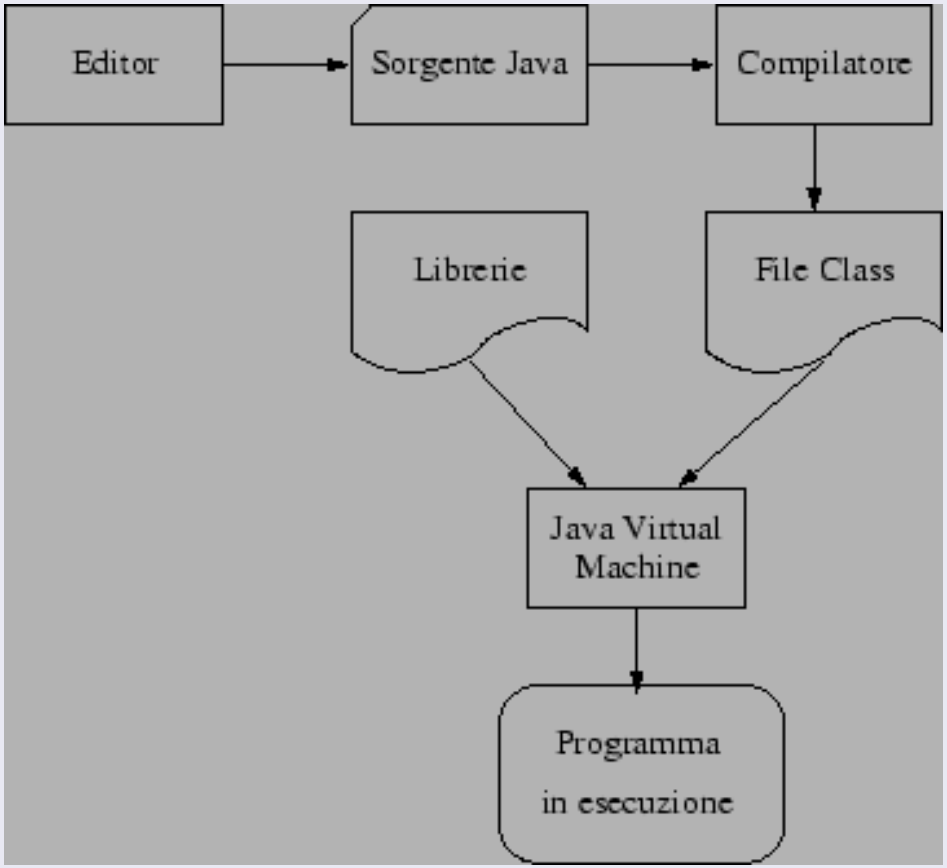
## Sintattici, Semantici e Logici

- Sintattici: violazione delle regole sintattiche del linguaggio
  - esempio: System.out.println(...) - manca il ';'
  - sono individuati dal compilatore
- Semantici, impossibilità di assegnare un significato ad un'istruzione
  - esempio: Sistem.out.println(...); - errore di ortografia nella parola System
  - a volte sono individuati dal compilatore (errori di semantica statica), altre volte sono individuati a tempo di esecuzione (errori di semantica dinamica)
- Logici, relativi alle funzionalità realizzate dal programma (differenti da quelle desiderate)
  - esempio: System.out.println(...e non saranno gli ultm!!!); : la stringa da stampare non è corretta
  - individuati solo analizzando o eseguendo test di verifica del programma

# Dal codice sorgente all'esecuzione



## Riassunto

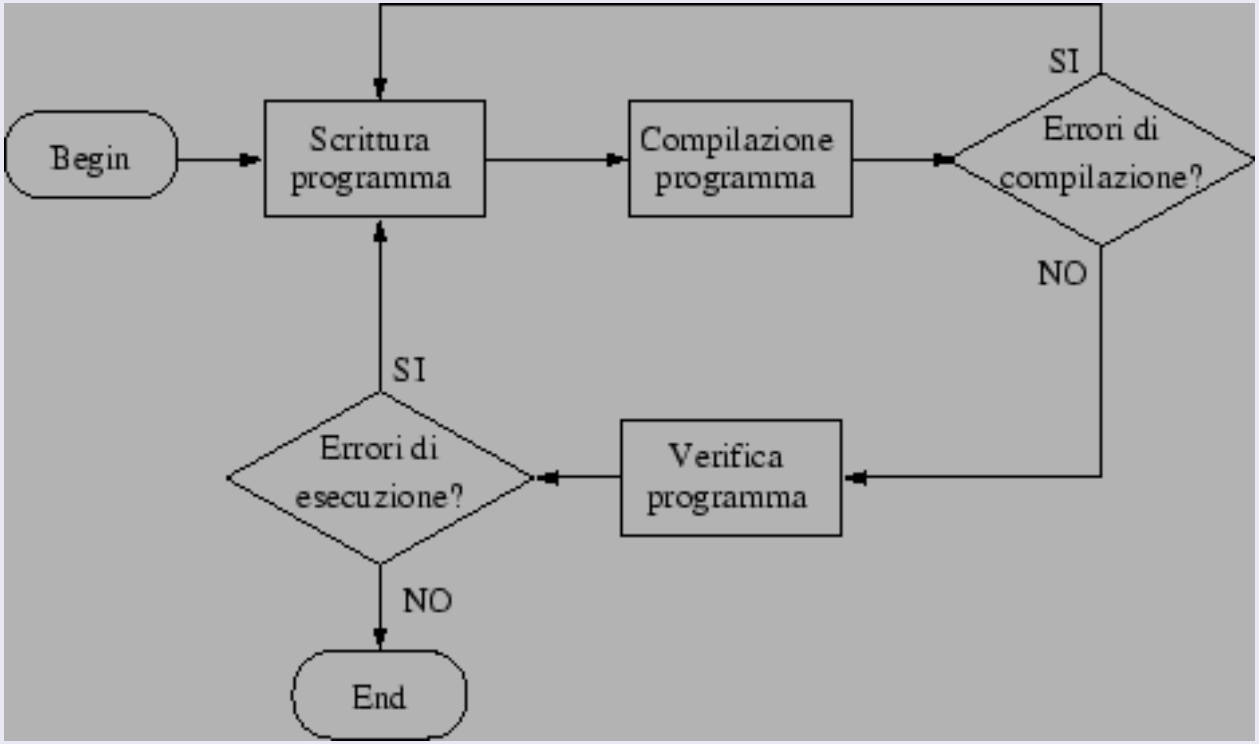



**Autore: Alessandro Farinelli**
**Corso di Laboratorio di Programmazione II**
**http://profs.sci.univr.it/~farinelli/pubs/publications.html**

# Ciclo edita compila esegui



## Riassunto



# Java API



## Abstract Programming Interface

- documentazione per Java
- Java API
- documentazioni di classi, metodi e package

# Oggetti e Metodi

# Scrivere un programma in Java



## Example (Un semplice programma java)

```
/*
* Un semplice Programma in Java
*/
/*Questo programma DEVE essere salvato come
Semplice.java
(maiuscole contano) */
public class Semplice{
    public static void main(String[] args){
        //stampa
        System.out.print("Questo e' un  semplice");
        //stampa e va a capo
        System.out.println("programma in JAVA");
    }
}
```

# Formato e commenti in Java



## Formato

- Le parole chiave sono separate da spazi (*class* oppure *void*)
- Spazi e linee non hanno alcun effetto sull'esecuzione del programma
- Andare a capo equivale ad uno spazio

## Commenti

- Possiamo annotare il programma con dei commenti
- I commenti sono fondamentali per la programmazione (non hanno effetto sull'esecuzione)
- //commento su una linea
- /*<testo>*/ <testo> puo' contenere qualsiasi carattere (anche andare a capo)

## Altre note sui programmi

- Tutti i programmi sono collezioni di classi in Java
- il metodo *main* indica dove inizia l'esecuzione del programma
- *System* e' una classe predefinita in Java
- *System.out* indica un *riferimento* ad una *oggetto* istanza della classe *PrintStream*
- *System.out* e' definito nella classe *System* e gestisce l'*output di sistema*
- *println(...)* e *print(...)* sono due metodi della classe *PrintStream*
- *System.out.println(...)* invoca il metodo *println(...)* sull'oggetto *System.out* (mandiamo un *messaggio* all'oggetto)

# Invocazione di metodi in Java



## Sintassi

- *rifOggetto.nomeMetodo(parametri)*
- Esempio *System.out.println(test)*
- Oggetto: *System.out*; Nome metodo: *println()*; Parametri: *test*
- L'invocazione di un metodo può anche restituire dei valori

# Oggetti in Java



## Utilizzare Oggetti

- Oggetti: entità manipolate dai programmi
- Ogni oggetto è istanza di una classe
- Una classe è costituita da un insieme di oggetti con le stesse caratteristiche
- La classe di appartenenza determina le operazioni eseguibili sull'oggetto

## Example (Utilizzo oggetti e metodi)

```
System.out.println("ciao") //OK
System.out.toUpperCase(); //ERRORE
```

# La classe String



## Note sulla classe String

- Oggetti di tipo *String* rappresentano sequenze di caratteri.
- vedere le API per i metodi disponibili
- Espressioni tra doppi apici (e.g., Ciao) rappresentano dei *riferimenti* ad oggetti di tipo String (detti letterali).
- Questi riferimenti sono chiamati **costanti String**
- Sono dei riferimenti:
  - possono essere passati come parametro
  - possono essere usati per invocare metodi (della classe String)

## Example (Utilizzo oggetti e metodi)

```
System.out.println("ciao".toUpperCase())
```

**Autore: Alessandro Farinelli**
**Corso di Laboratorio di Programmazione II**
**http://profs.sci.univr.it/~farinelli/pubs/publications.html**

# Signature e intestazione



## Signature

- signature: identifica un metodo
- nome metodo e numero, ordine e tipo dei parametri
- Nome dei parametri non e' significativo

## Intestazione

- signature + tipo di ritorno

## Note

- *void* il metodo non ritorna nulla (compie solo operazioni)
- Due metodi (della stessa classe) possono avere lo stesso nome se hanno signature differenti
- Due metodi che hanno stesso nome ma signature differenti si dicono *overloaded*

# Signature e intestazione: Esempio



## Example (Signature, Intestazione e overloading)

```
//ritorno String, nessun parametro
String toUpperCase()
//nessun ritorno, parametro: stringa da stampare
void println(String s)
//indice di occorrenza per la stringa str
int indexOf(String str)
//indice di occorrenza per la stringa str da
//fromIndex in poi OVERLOADING!
int indexOf(String str, int fromIndex)
```

# Parametri e risultato



## Cosa sono i parametri

- Parametri: argomenti usati in un metodo per eseguire le operazioni richieste
- Il ritorno di un metodo (se presente) viene passato al chiamante

## Example (Parametri e risultato)

```
//stampa il parametro passato al metodo
System.out.println("ciao!")
//la string "CIAO!" viene restituita dal
//metodo toUpperCase() e passata al metodo println()
System.out.println("ciao!".toUpperCase())
```

# Valutazione dei parametri nell' invocazione di un metodo



## Come vengono valutati i parametri

In generale nell'invocazione di un metodo i parametri potrebbero essere ottenuti tramite l'invocazione di altri metodi.
La valutazione dei parametri avviene da sinistra verso destra

## Example (Esempio valutazione)

```
System.out.println("ja".toUpperCase().
 concat("va".toUpperCase()));
System.out.println("ja".concat("v").
 concat("a").toUpperCase());
```

# Metodi statici



## Cosa sono i metodi statici

I metodi statici non hanno un oggetto di invocazione.
Sintassi:

```
NomeClasse.nomeMetodo(parametri)
```

*NomeClasse* NON e' un oggetto ma il nome della classe in cui il metodo e' definito
Specifichiamo *NomeClasse* perche' i metodi hanno un nome locale alla classe
Tutte le operazioni di un metodo statico usano solo i parametri passati al metodo

# Metodi statici: Esempio



## Example (Metodi statici e non)

```
//max e' un metodo statico della classe Math
//restituisce il max tra i parametri
System.out.println(Math.max(5,6));
//toUpperCase() e' un metodo NON statico della
//classe String. Il metodo non ha parametri
//perche' la stringa in input e' l'oggetto di
//invocazione
System.out.println("ciao".toUpperCase());
```

# Esercizi



## Metodi e Oggetti

I due esercizi devono essere svolti senza usare variabili, ed assumendo di non saper contare il numero di caratteri delle stringhe.

M1 Scrivere un programma Java che crei un oggetto String che rappresenta un nome e stampi il primo e l'ultimo carattere della stringa. *Soluzione*: Nome.java

M2 Scrivere un programma Java che date due stringhe di caratteri minuscoli stampi le due stringhe con il primo carattere maiuscolo. *Soluzione*: PrimaMaiuscola.java

# Variabili



## Uso delle variabili

Variabile: locazione di memoria che puo' essere usata per memorizzare il riferimento ad un oggetto

```
System.out.println("john".charAt("john".length()-1));
...
String nome = John
System.out.println(nome.charAt(nome.length()-1));
/***************/
System.out.println("john".toUpperCase());
System.out.println("john".toUpperCase());
...
String nome = "john".toUpperCase();
System.out.println(nome);
System.out.println(nome);
```

# Variabili: Caratteristiche



## Caratteristiche principali delle variabili

- **Nome**: serve ad identificare ad una variabile (e.g., la variabile *nome*)
  - nome: sequenza di lettere e cifre ed il carattere '_', non puo' iniziare con una cifra
  - puo' avere qualsiasi lunghezza
  - maiuscole e minuscole sono differenti
  - non puo' essere una parola chiave (**if, else, while,class,..**)
- **Tipo**: tipo di dato che puo' essere memorizzato (e.g., *String*)
- **Valore**: il dato che la variabile memorizza (e.g., *john*)
- **Indirizzo**: locazione di memoria che contiene il valore
  - In Java non c'e' alcun modo di conoscere la locazione di memoria di una variabile (scelta di progettazione).

Nome, tipo ed indirizzo non cambiano mai, il valore si.

# Variabili: Dichiarazione



## Dichiarazione di Variabili

- **Sintassi**: *tipo nomeVariabile*
- dopo la dichiarazione la variabile puo' essere usata nel blocco di codice relativo alla dichiarazione
- E.g.:

```
String line;
String line1, line2, line3;
```

# Variabili: Assegnazione



## Assegnare valori ad una variabile

- **Sintassi**: *nomeVariabile* = *espressione*
- la variablile *nomeVariabile* assume il valore del risultato di *espressione*
- il tipo della variabile deve essere in accordo con il tipo del valore del risultato dell'espressione.

## Example (Esempi di assegnazione)

```
//assegnazione corretta
String s = "ja";
//assegnazione corretta
s = s.concat("va");
//errore
s = 3
```

# Variabili: Inizializzazione



## Inizializzare una variabile

- Inizializzare una variabile vuol dire associare un valore alla variabile per la prima volta;
- Una variabile non ionizializzata non puo' essere usata (valore non definito);
- Java restituisce in questi casi un errore a tempo di compilazione.

## Example (Esempi di inizializzazione)

```
String line;
System.out.println(line); //ERRORE
...
String line = "java";
System.out.println(line); //OK
```

# Riferimenti e Oggetti



## Variabili che si riferiscono ad oggetti

- Una variabile in Java non contiene **mai** un oggetto ma un **riferimento** ad un oggetto;
- Due variabili possono contenere il riferimento ad uno **stesso** oggetto;
- Gli oggetti vengono allocati in memoria in maniera indipendente dalle variabili;
  - i letterali vengono allocati in fase di compilazione
  - gli altri oggetti vengono allocati tramite istruzione esplicita: *new NomeClasse()*
- possono assumere un valore speciale: *null*, diverso da non essere inizializzate.

# Esempi di variabili riferimenti a Oggetti



**Example (Uso di variabili riferimento a oggetti)**

```
String s = "ciao"
String t = s
//t ed s si riferiscono allo stesso oggetto ("ciao")
String v = new String()
//v contiene la string vuota
//(vedremo di piu' su questo in seguito)
```

# Esempio di manipolazione stringhe





## Example (Iniziali)

```
public class Iniziali{
    public static void main(String[] args){
    String nome = "John";
    String cognome = "Bonham";
    System.out.println(nome.substring(0,1).
     concat(cognome.substring(0,1)));
    }
}
```

# Esempio di manipolazione stringhe



## Example (Iniziali con uso di +)

```
public class Iniziali{
    public static void main(String[] args){
    String nome = "John";
    String cognome = "Bonham";
    System.out.println(nome.substring(0,1)+
     cognome.substring(0,1));
    }
}
```

Vedere file Iniziali

# Invocazione del costruttore



## Creazione di oggetti tramite il costruttore

- **Sintassi**: *new NomeClasse(parametri)*
- new parola chiave per creare un oggetto
- *NomeClasse(parametri)* il metodo costruttore
  - restituisce un riferimento all'oggetto creato
  - inizializza l'oggetto usando i parametri

## Example (Costruzione oggetti)

```
//crea una stringa che contiene "ciao"
String s = new String("ciao");
//crea una stringa vuota (0 caratteri)
String s = new String();
```

I due costruttori hanno stesso nome ma signature diverse: Overloading

# Accessibilita' ad Oggetti



## Oggetti senza riferimenti

- Un oggetto esiste a prescindere dal fatto che ci sia un riferimento associato ad esso
- Un oggetto senza riferimento non potra' mai essere usato
- In Java possiamo deallocare esplicitamente memoria
- **Garbage Collector**: recupero della memoria automatico della JVM
- Possiamo forzare l'esecuzione del Garbage Collector (*System.gc()*)

## Example (Utilizzo oggetti e metodi)

```
String s = new String("ciao");
String t = new String("lost");
String t = s; //perso riferimento a "lost"
```

# Oggetti Immutabili



## La classe String

- Per scelta implementativa gli oggetti di tipo *String* una volta creati non possono mai essere modificati.
- I metodi che debbono modificare la sequenza di caratteri di oggetti String restituiscono *sempre* un *nuovo oggetto*

## Example (Oggetti Immutabili)

```
String s = "ciao";
System.out.println(s.toUpperCase());
System.out.println(s);
```

# Oggetti Mutabili



## La classe StringBuffer

- Dispone di metodi che modificano l'oggetto di invocazione

## Example (Oggetti Mutabili)

```
StringBuffer sb = "ciao";
System.out.println(sb.appen("!"));
System.out.println(sb);
```

# Side Effect



## metodi che effettuano side effect

- Side-effect (effetto collaterale), modificare dello stato di un oggetto
- Necessario ma da considerare con cura (potrei avere modifiche non attese)

## Example (Side Effect)

```
StringBuffer sb1 = "ciao";
StringBuffer sb2 = sb1;
sb1.appen("!"));
System.out.println(sb1);
System.out.println(sb2);
```

# Input da tastiera



## Leggera input da tastiera

- Importare la libreria Java.util.*
- `import Java.util.*;`
- Inizializzare lo stream di ingresso
- `Scanner sc = new Scanner(System.in)`
- Per leggere un dato (String) da tastiera
- `String s = sc.nextLine();`
- Vedi file: EsempioInput.java

# Esercizi



## Variabili I

V1 Considerare il file Errato.java che contiene alcuni errori di compilazione. Eliminare tutti gli errori di compilazione *Soluzione*: Errato.sol

V2 Scrivere un programma Java che date due stringhe di caratteri minuscoli, rappresentate tramite StringBuffer, stampi le due stringhe con il primo carattere maiuscolo. *Soluzione*: PrimaMaiuscolaSB.java

V3 Prende in input una stringa e stampa la stringa che ha l'ultima e la prima lettera invertite *Soluzione*: PrimaUltima.java

# Esercizi



## Variabili I

V4 Scrivere una classe ConcatenaStringhe che legge da input 4 stringhe, le concatena e stampa la stringa risultante. Usare una variabile String. *Soluzione*: ConcatenaStringhe.java

V5 Riscrivere la classe ConcatenaStringhe usando una variabile StringBuffer e non String. *Soluzione*: ConcatenaStringheSB.java