Introduzione a String

String → oggetti creabili nel seguente modo `String s = new String("abcdef");`

Ci sono sottili differenze oppure

`String s = "abcdef";`

Immutabilità

Le stringhe, come i tipi Wrapper, sono immutabili (contenuto non modificabile).

Tuttavia, il riferimento può cambiare.

String pool

La JVM riserva un area di memoria per le stringhe per questioni di efficienza, la String constant pool.

Il compiler controlla se una stringa è già presente:

- È presente => viene interpretato come riferimento già esistente;
- Non è presente => viene creato un nuovo oggetto e aggiunto alla pool;

Funziona solo perché sono immutabili

Qual è quindi la differenza tra i due enunciati?

```
String s = "abcdef";
```

```
String s = new String("abcdef");
```

Obbliga la creazione di un oggetto String oltre quello della JVM

StringBuffer e StringBuilder

→ Thread-safe

Vengono usati quando c'è bisogno di manipolare la stringa

```
StringBuffer s = new StringBuffer("Walter");
s.append("␣White");
System.out.println(s); // cosa stampa?

String contenuto = s.toString();
```

Attenzione:

```
StringBuffer s = "abc";
  // Illegale: String e StringBuffer
  // non sono auto-convertibili!

StringBuffer s = (StringBuffer) "abc";
  // Illegale: neanche con cast!
```

Uguaglianza

Attenzione → String fa override di .equals per confrontare il contenuto degli oggetti.

StringBuilder/buffer ereditano quello di Object (funzia come == e confronta i riferimenti)

Tutti questi tipi sono final e non possono essere specializzati.

Garbage Collection

Visto che la memoria non può essere gestita, la GC automatizza il processo.
Un data object **è eleggibile** per la GC se non è più accessibile dal programma

Esempio:

```
public static void main(String[] args) {
   StringBuffer sb = new StringBuffer("Ciao");
   System.out.println(sb);
   sb = null; // Viene perso il riferimento
   // ora l'oggetto StringBuffer e' eleggibile per la GC
}
```

Nota: tranne casi eccezionali, le stringhe nello string pool non sono mai eleggibili per la GC

Algoritmo di GC

Mark-and-sweep -> **Fase mark**: dallo stack e dalla regione statica, marca tutti gli oggetti
simile alla visita
in un grafo
(i nodi rappresentano
la regione statica)

Fase sweep: esplora la regione dinamica e rilascia i data object
non marcati

```
public static void main(String[] args) {
   StringBuffer s1 = new StringBuffer("Ciao");
   StringBuffer s2 = new StringBuffer("Addio");
   System.out.println(s1);
   // l'oggetto riferito da s1 non e' ancora
   // eleggibile per GC
   s1 = s2;
   // qui e' eleggibile
   ...
}
```

Simulare la procedure di GC mark-and-sweep nel punto segnato nel seguente programma:

```
class Employee {
   private String name;
   private Employee boss;
   public final static Employee CEO = new Employee("Gustavo", null);
   ...

   public static void main(String[] args) {
      Employee w = new Employee("Walter", CEO);
      f();
   }
   public static void f() {
      Employee j = new Employee("Jesse", CEO),
               p = new Employee("Pete", j);
      ArrayList<Employee> l = new ArrayList<>();
      l.add(j);
      j = null;
      p = null;
      // simulare la GC a questo punto
   }
}
```

Memory layout di questo codice