Metodologie di Programmazione

Alessia Cassetta

March 2024

Indice

1 Introduzione

Tipo	Intervallo	Dimensione
double	parte intera: ±10308, parte frazionaria: circa 15 cifre decimali significative	8 byte
long	-9223372036854775808…9223372036854775807	8 byte
int	-2147483648…2147483647	4 byte
float	parte intera: ±1038, parte frazionaria: circa 7 cifre decimali significative	4 byte
boolean	true, false	1 byte
char	Rappresenta tutti i caratteri codificati con Unicode	2 byte
short	-32768…32767	2 byte
byte	-128…127	1 byte

Figura 1: **Python VS Java**

1.1 La programmazione e Generalità

Molto più semplice che apprendere una lingua straniera. Poche parole chiave da imparare. Portabilità del bagaglio: la maggior parte di ciò che imparate in Java, potete applicarlo in Python, C, in C++ o nella maggior parte degli altri linguaggi di programmazione.

E' fondamentale apprendere a programmare (a oggetti), non (solo) apprendere Java:

- sapere quali dettagli sono rilevanti in quale situazione
- modellare la realtà mediante l'utilizzo di oggetti
- astrarre e generalizzare per poter riutilizzare

Polimorfismo: E' possibile utilizzare la classe base, senza dover conoscere necessariamente la classe specifica di un oggetto. Permette di scrivere codice che non dipende dalla classe specifica. Posso aggiungere nuove sottoclassi anche in seguito!

1.2 Perché Java come linguaggio di riferimento?

Un linguaggio di programmazione potente, orientato agli oggetti. Creato da Sun Microsystems (ora Oracle).

Precursori: Smalltalk (fine ‘70), C++ (inizio ‘80). Costruito per essere “sicuro”, cross-platform e internazionale.

Continuamente aggiornato con nuove feature e librerie.

Multithreaded: Supporta nativamente programmi che gestiscono attività eseguite in contemporanea (thread). Facilita la costruzione di applicazioni interattive.

Interpretato e compilato: il bytecode è tradotto “al volo” in istruzioni macchina native. Rende più veloce e snello il processo di sviluppo.

2 Tipi di dato di base (o primitivi)

I tipi di dati di base sono built-in, ovvero sono predefiniti nel linguaggio. È fondamentale essere a conoscenza di quali siano i tipi di base e quali non lo siano. Per ragioni di efficienza e di allocazione della memoria.

Un tipo di dati è un insieme di valori e di operazioni definite su tali valori: Interi (es. 27), Reali (es. 27.5) Booleani (true o false), Caratteri (es. ‘a’), Stringhe (es. “questa non è una stringa”).Una variabile è un nome usato per riferirsi a un valore di un tipo di dati (es. contatore).

Tipo	Intervallo	Dimensione
double	parte intera: ±10308, parte frazionaria: circa 15 cifre decimali significative	8 byte
long	-9223372036854775808…9223372036854775807	8 byte
int	-2147483648…2147483647	4 byte
float	parte intera: ±1038, parte frazionaria: circa 7 cifre decimali significative	4 byte
boolean	true, false	1 byte
char	Rappresenta tutti i caratteri codificati con Unicode	2 byte
short	-32768…32767	2 byte
byte	-128…127	1 byte

Figura 2: Enter Caption

Una variabile è creata mediante una dichiarazione:

```
int contatore;
```

Il valore viene assegnato a una variabile mediante una assegnazione:

```
contatore = 0;
```

Un'istruzione può includere una dichiarazione e una assegnazione allo stesso tempo:

```
int contatore = 0;
```

In Java siamo costretti a specificare il tipo della variabile.
Questo tipo non può più cambiare (ovvero è statico). Tuttavia, esistono alcuni elementi di dinamicità (che vedremo più avanti). Non posso utilizzare una variabile senza prima dichiararla e non posso assegnare a una variabile tipi incompatibili tra loro.

Il nome assegnato a una variabile è detto identificatore, ovvero una sequenza di lettere, cifre, _ e $ la prima delle quali non è una cifra. Gli identificatori sono case-sensitive. Non possono essere utilizzate alcune parole riservate (es. public, static, int, double, ecc.). Si utilizza la notazione a cammello (Camel case). Le variabili devono avere un **nome sensato**.

Esempio di Codice

```
int a, b;
a = 5;
b = a + 10;
int c = a+b;
a = c - 3;
```

2.1 I letterali

Un **letterale** è una rappresentazione a livello di codice sorgente del valore di un tipo di dati. Ad esempio: 27 o -32 sono letterali per gli interi; 3.14 è un letterale per i double; true o false sono gli unici due letterali per il tipo booleano; "Ciao, mondo" è un letterale per il tipo String.

2.2 Distinguere tra costanti intere e in virgola mobile

Interi:

- Le costanti di tipo int sono semplicemente numeri nell'intervallo [-2, +2] miliardi circa.
- Le costanti di tipo long vengono specificate con il suffisso l o L (ad esempio, 1000000000000L).

Numeri in virgola mobile:

- Le costanti di tipo double sono semplicemente numeri con la virgola (punto).
- Le costanti di tipo float hanno il suffisso f o F (ad esempio, 10.5f).

In tutti i casi si può utilizzare il trattino basso (_) per separare le cifre (es. 100_000 per indicare 100000, 1_234 per indicare 1234 ecc.). Si può ottenere un intero da una rappresentazione binaria anteponendo 0b alla stringa binaria (ad esempio, 0b101 vale 5).

2.3 Altre generalità

Espressioni: è un letterale, una variabile o una sequenza di operazioni su letterali e/o variabili che producono un valore.

Assegnazione di un'espressione a una variabile: supponiamo di voler effettuare l'assegnazione:

```
1 c = a*2+b
```

1) Calcola il valore dell'espressione destra (5*2+15).
2) Assegna il valore (25) alla variabile di destinazione (c).

2.4 Conversioni di tipo

Conversione Esplicita: utilizzando un metodo che prende in ingresso un argomento di un tipo e restituisce un valore di un altro tipo. Integer.parseInt(), Double.parseDouble(), Math.round(), Math.floor(), Math.ceil() ecc...

Cast esplicito: anteponendo il tipo desiderato tra parentesi **(int)2.71828** produce un intero di valore 2. Se il tipo di partenza è più preciso (es. double), le informazioni aggiuntive vengono eliminate nel modo più ragionevole (es. da double a int viene eliminata la parte frazionaria).

Cast implicito: se il tipo di partenza è meno preciso, Java converte automaticamente il valore al tipo più preciso: double d = 2; **Attenzione: la somma di due caratteri dà un intero!**

2.4.1 Regole per il cast implicito

Il cast implicito avviene in fase di assegnazione:

- byte, short e char possono essere promossi a int
- int può essere promosso a long
- float può essere promosso a double

O in fase di calcolo di un'espressione: se uno dei due operandi è double, l'intera espressione è promossa a double. Altrimenti, se uno dei due operandi è float, l'intera espressione è promossa a float.

3 Classi e Oggetti

Possono:

- Modellare gli oggetti del mondo reale
- Rappresentare oggetti grafici
- Rappresentare entità software (file, immagini, eventi, ecc.).
- Rappresentare concetti astratti (regole di un gioco, posizione di un giocatore).
- Rappresentare stati di un processo, di esecuzione, ecc...

Una classe è un pezzo del codice sorgente di un programma che describe un particolare tipo di oggetti.
Le classi vengono definite dal programmatore e forniscono un prototipo astratto per gli oggetti di un particolare tipo. Ne definisce la struttura in termini di **attributi** (stato degli oggetti) e di **metodi** (comportamenti) degli oggetti.

La **classe** è definita mediante parte del codice sorgente del programma ed è implementata dal programmatore.
Specifica la struttura degli attributi dei suoi oggetti e il comportamento dei suoi oggetti tramite metodi.

Un **oggetto** è un entità della classe programmata in esecuzione e viene creata quando un programma ”gira” nel main o in un altro metodo.
Contiene specifici valori dei campi che possono cambiare durante l'esecuzione. Inoltre, si comporta nel modo prescritto dalla classe quando il metodo corrispondente viene chiamato a tempo di esecuzione.

3.1 Classi e file sorgenti

Ogni classe è memorizzata in un file separato. Il nome del file deve essere lo stesso della classe con estensione .java
I nomi di classe iniziano sempre con una maiuscola e sono case-sensitive.

3.1.1 Librerie

I programmi Java normalmente non sono scritti da zero ma esistono migliaia di classi di librerie per ogni esigenza. Le classi sono organizzate in **package**.

3.2 Attributi e Metodi

Un **attributo** costituisce la memoria privata di un oggetto e ha un tipo di dato. Ogni campo ha un nome fornito dal programmatore.

Ad esempio:

```
public class Hotel{
    private int k;
    private String nome;
    private int numeroStanze;
}
```

I **metodi** sono tipicamente pubblici, ovvero visibili a tutti. Il nome del metodo per convenzione inizia con una lettera minuscola, mmentre le parole seguenti iniziano con una lettera maiuscola (camelCase).

Ad esempio:

```
public String testMethod(paramtee){
    //instructions
}
```

In questo modo ho definito una classe che restituisce una stringa ed ottiene in input dei parametri.

Come vengono chiamati i metodi:

```
public void reset(){value = 0;}
public void reset(int valueIn){value = valueIn;}
static public void main(String[], args){
    Counter contatore1 = new Counter(42);
    contatore1.count();
    contatore1.reset();   // chiama il primo metodo reset
    contatore1.reset(10); //chiama il secondo
}
```

Quando troviamo la parola **void** nell'intestazione del metodo, indica che il metodo non restituisce alcun valore.

3.2.1 Campi di classe: la parola chiave static

I campi di una classe possono essere dichiarati static. Un campo static è relativo all'intera classe, NON al singolo oggetto istanziato. Un campo static esiste in una sola locazione di memoria, allocata prima di qualsiasi oggetto della classe in una zona speciale di memoria nativa chiamata MetaSpace.

- **Espressioni**: è un letterale, una variabile o una sequenza di operazioni su letterali e/o variabili che producono un valore.

- **Assegnazione di un'espressione a una variabile**: supponiamo di voler effettuare l'assegnazione:

```
1     c = a*2+b
2
```

  1) Calcola il valore dell'espressione destra (5*2+15).
  2) Assegna il valore (25) alla variabile di destinazione (c).

Viceversa, per ogni campo non static esiste una locazione di memoria per ogni oggetto, allocata a seguito dell'istruzione new.

3.3 Metodi Statici

I seguenti punti illustrano le caratteristiche dei metodi statici:

- Un metodo statico è un metodo che appartiene alla classe, non all'oggetto.
- Un metodo statico può essere chiamato anche se non esiste alcun oggetto della classe.
- Un metodo statico non può accedere a campi non statici.
- Un metodo statico può essere chiamato direttamente dalla classe, senza creare un oggetto.

Esempio:

```
public class MyClass {
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating
   objects");
  }
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating
   objects");
  }
  public static void main(String[] args) {
    myStaticMethod();
    // Call the static method
    MyClass myObj = new
    MyClass(); // Create an object of MyClass
    myObj.myPublicMethod();
    // Call the public method
    }
}
```

3.4 Input

- Lettura dell'input da console
  - Si effettua con la classe java.util.Scanner
  - Costruita passando al costruttore lo stream di input (System.in di tipo java.io.InputStream)

```
import java.util.Scanner;
public class InputExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        System.out.println("Hello, " + name + "! You are " + age + "
   years old.");

        scanner.close();
    }
}
```

3.5 Costruttori

Servono ad inizializzare gli attributi dell'oggetto e posseggono lo stesso nome della classe. Non hanno valori di uscita. Non è obbligatorio specificare un costruttore dato che Java crea per ogni classe un costruttore di default "vuoto".

```
public Counter(int valueIn){
    value = valueIn;
}
static public void main(String[], args){
    Counter contatore1 = new Counter(42);
}
```

I **campi** sono variabili dell'oggetto. Sono visibili almeno all'interno di tutti gli oggetti della stessa classe. Esistono per tutta la vita di un oggetto.

Le **variabili locali** sono variabili definite all'interno di un metodo. Come parametri del metodo o all'interno del corpo del metodo. Esistono dal momento in cui sono definite fino al termine dell'esecuzione della chiamata al metodo in questione.

3.6 Incapsulamento

Le parole **public** e **private** sono usate per specificare l'accesso ai campi e ai metodi. Il processo che nasconde i dettagli realizzativi di un oggetto e mostra solo le funzionalità è chiamato **incapsulamento**.
Utilizziamo l'incapsulamento per proteggere i campi da modifiche accidentali. Non è necessario sapere tutto, soprattutto molti inutili dettagli. Esso facilita il lavoro di gruppo e aiuta a rilevare errori più facilmente in quanto si può concentrare solo su una parte del codice.
Una classe interagisce con le altre classi mediante i metodi pubblici.

3.6.1 Accesso a campi e metodi (inclusi i costruttori)

I campi e i metodi possono essere **pubblici**, **privati** o **protetti**.

I metodi di una classe possono accedere ai metodi privati e pubblici della stessa classe e possono chiamare metodi pubblici di altre classi.

3.7 UML

L'**UML** (Unified Modeling Language) è un linguaggio di modellazione visuale utilizzato per descrivere, visualizzare, specificare e documentare i sistemi software. È uno standard industriale ampiamente utilizzato nel campo dello sviluppo software. L'UML fornisce una serie di diagrammi che consentono di rappresentare diversi aspetti di un sistema software, come la struttura, il comportamento, le interazioni e le relazioni tra le varie componenti del sistema. Questi diagrammi aiutano a comunicare in modo chiaro e conciso le informazioni sul sistema tra gli sviluppatori, gli stakeholder e gli altri membri del team.

- Diagramma dei casi d'uso: rappresenta le funzionalità del sistema e le interazioni tra gli attori (utenti o altri sistemi) e il sistema stesso.
- Diagramma delle classi: mostra le classi del sistema, i loro attributi, i metodi e le relazioni tra di esse.
- Diagramma delle sequenze: illustra l'interazione tra gli oggetti del sistema nel tempo, mostrando l'ordine delle operazioni e le chiamate di metodo.
- Diagramma delle attività: descrive il flusso di lavoro o il processo di un'attività, mostrando le azioni, le decisioni e le condizioni.
- Diagramma dei componenti: rappresenta le componenti del sistema e le loro dipendenze.
- Diagramma dei pacchetti: organizza le classi e i componenti in gruppi logici o moduli.

L'UML è uno strumento potente per la progettazione e l'analisi dei sistemi software. Aiuta a visualizzare e comprendere la struttura e il comportamento del sistema, facilitando la comunicazione tra gli sviluppatori e consentendo una migliore collaborazione nel processo di sviluppo del software. I diagrammi UML utilizzano una serie di simboli e notazioni per rappresentare le varie componenti di un sistema software. Ecco alcuni dei simboli più comuni utilizzati nei diagrammi UML:

- +: Indica che un metodo o un campo è pubblico. Questo significa che può essere accessibile da qualsiasi altra classe.
- -: Indica che un metodo o un campo è privato. Questo significa che può essere accessibile solo all'interno della stessa classe.
- #: Indica che un metodo o un campo è protetto. Questo significa che può essere accessibile all'interno della stessa classe e dalle sue sottoclassi.
- : Indica che un metodo o un campo ha visibilità package. Questo significa che può essere accessibile dalle classi nello stesso pacchetto.

3.8 Le Stringhe

Una classe fondamentale, perché è relativa a un tipo di dato i cui letterali sono parte della sintassi del linguaggio e per il quale il significato dell'operatore + è ridefinito. Non richiede l'importazione perché appartiene al package "speciale" java.lang. La classe String è dotata del metodo length()

Esempio:

```
    String s = "Ciao";
    System.out.println(s.length());
```

Stamperà il valore 4, che è pari al numero di caratteri di cui è costituita la stringa s

Con i metodi toLowerCase() e toUpperCase() si ottiene un'altra stringa tutta minuscola o maiuscola, rispettivamente. La stringa su cui viene invocato il metodo non viene modificata

Esempio:

```
String min = "Ciao".toLowerCase(); // "ciao"
String max = "Ciao".toUpperCase(); // "CIAO"
String ariMin = max.toLowerCase(); // "ciao"
```

E' possibile ottenere il k-esimo carattere di una stringa con il metodo `charAt`

- Importante:
  - il primo carattere è in posizione 0
  - l'ultimo carattere è in posizione `stringa.length()-1`
- Esempio:

```
    String s = "ciao";
    System.out.println(s.charAt(2));

```

Stamperà il carattere 'a' (si noti che `charAt` restituisce un carattere (tipo `char`), non una stringa)

3.9 Ottenere una sottostringa

- È possibile ottenere una sottostringa di una stringa con il metodo `substring(startIndex, endIndex)`, dove:
  - `startIndex` è l'indice di partenza della sottostringa
  - `endIndex` è l'indice successivo all'ultimo carattere della sottostringa

- Ad esempio:

```
1     String s = "ciao";
2     System.out.println(s.substring(1, 3));
3
```

  Stamperà la stringa ”ia”, dalla posizione 1 (’i’) alla posizione 3 (’o’) esclusa

3.10 Concatenare stringhe

- La concatenazione tra due stringhe può essere effettuata con l’operatore ”+” oppure mediante il metodo `concat(s)`
- Tuttavia, se si devono concatenare parecchie stringhe, è bene utilizzare la classe `StringBuilder`, dotata dei metodi `append(String s)` e `insert(int posizione, String s)`

3.11 Cercare in una stringa

- Si può cercare la (prima) posizione di un carattere `c` con `indexOf(c)`, restituisce -1 se il carattere non è presente
- È possibile anche cercare la prima posizione di una sottostringa con `indexOf(s)` dove `s` è di tipo `String`

3.12 Sostituire caratteri e sottostringhe

- Con il metodo `replace` è possibile sostituire tutte le occorrenze di un carattere o di una stringa all’interno di una stringa

3.13 Confrontare stringhe

- Le stringhe vanno SEMPRE confrontate con il metodo `equals`
- L’operatore `==` confronta il riferimento (indirizzo in memoria), mentre `equals` confronta la stringa carattere per carattere

3.14 Spezzare le stringhe

- Il metodo `split` restituisce un array di sottostringhe separate da un’espressione regolare