Programmazione 05/10/22

Costruttori

- Stesso nome della classe
- Nessun tipo di ritorno
- Possono avere parametri
- Ce ne possono essere diversi per la stessa classe
- Chiamati automaticamente in fase di istanziazione

```
// Esempio
public class Punto {
    public Punto(int a, int b){
        x = a;
        y = b;
    }
    public int x;
    public int y;
}

// main
Punto punto1; // dichiarazione
punto1 = new Punto(5, 6); // istanza
punto1 = new Punto(); // Errore ??? (richiede argomenti)
```

```
// Esempio con più costruttori
public class Punto{
    public Punto(){
        // Questo costruttore non fa niente
    }
    public Punto(int a, int b){
        x = a;
        y = b;
    }
    public int x;
    public int y;
}
```

Costruttore di default

- Inserito automaticamente dal compilatore
- Solo se il programmatore non ne ha fornito uno esplicitamente!

Packages

- Organizzare il codice in cartelle
- Il comando import
    - Evita di scrivere il nome del package completo (es. java.lang.String)
    - java.lang è sempre importato automaticamente dal compilatore
    - es: import java.util.ArrayList
- Come fa una classe a usare classi che si trovano in altre cartelle/package?
- Ruolo dell'IDE

Altri componenti

- Interfacce
    - Strumento utile per la progettazione
- Enumerazioni
- Annotazioni
- Moduli (J9)
    - es: java.base
    - Si usa per progettare programmi di dimensioni elevate
- Inizializzatori
- Classi innestate
- Espressioni lambda (J8)
    - "Funzioni senza nome"

Identificatori, tipi di dati e array

- Lo stile di codifica è molto analogo al linguaggio C. **Java è un linguaggio a schema libero.**
- Ovviamente la tabulazione non è necessaria (se vuoi criptare un programma)

- E' case sensitive come C
- I commenti sono analoghi a C ma c'è una novità:
    - Javadoc /** ... */
        - I commenti Javadoc messi in una determinata posizione generano documentazione in HTML.

Identificatori

- Nomi di metodi, classi, oggetti, variabili, costanti, ecc.
- Non possono coincidere con le parole chiave di Java
- Primo carattere A - Z, a - z, _ , $
- Secondo e successivi A - Z, a - z, _, $, 0 - 9

Convenzioni sui nomi

- Nomi significativi
- Classi
    - Usare il singolare
    - Iniziano con lettera maiuscola
    - Maiuscolo per separare le parole
- Variabili e metodi
    - Iniziano con lettera minuscola
    - Variabili -> sostantivi (es. numeroLati)
    - Metodi -> verbi (es. stampaPerimetro)
- Costanti
    - Tutte maiuscole
    - `_` per separare le parole (es: PI_GRECO)
- Package
    - Tutte minuscole (es: it.lacascia.prg)

Tipi di dati primitivi

- Soltanto 8 tipi:
    - `interi` : byte, short, int, long
    - `floating point` : float e double

    - `testuale` : char
    - `logico-booleano` : boolean
- Non ci sono gli unsigned come in C

Tipo	Intervallo di rappresentazione
byte	8 bit
short	16 bit
int	32 bit
long	64 bit

Letterali interi

- Decimali, binari, ottali, esadecimali
- Esempio:

```
byte b = 10; // notazione decimale: b vale 10
short s = 022; // notazione ottale: s vale 18
long l = 0x12acd; // notazione esadecimale: l vale 76493

int i = 1000000000;
// notazione decimale: i vale 1000000000

int n = 0b1010...0101
// notazione binaria: n vale - 1589272251
```

- `Nota: assegnazione fuori range -> errore di compilazione`

Promozioni di tipo

```
``` java
byte b = 200; // ERRORE

byte b = 50; // OK

byte b = 50 * 2; // OK

byte b = 50;
b = b * 2; // ERRORE
```
```

Anche se 100 può stare in una variabile di tipo byte potrebbe segnalare un errore!

Promozione automatica nelle espressioni aritmetiche

- Se uno degli operandi è `double` l'altro verrà convertito in `double`
- Se il più ampio (il tipo più grande degli operandi) degli operandi è `float` l'altro verrà convertito in `float`
- Se il più ampio degli operandi è `long` l'altro verrà convertito in `long`
- In tutti gli altri casi entrambi gli operandi sono convertiti in `int`

Casting esplicito

- b = (byte) (b * 2); // OK
- b = (byte)128; // OK ma b vale -128;
    - 128 è un int a 32 bit e viene troncato prendendo solo gli 8 bit meno significativi

`DA USARE CON CAUTELA E CONSAPEVOLEZZA!`

Troncamento in caso di overflow

```java
int a = 2147483647; // massimo valore per un int
int b = 1;
int risultato = a + b;
```

- La variabile risultato vale -2147483648
- Possibile soluzione : usare `long`
    - `long` risultato = a+b; // `Non corretto`
    - Si perde b perché l'operazione a + b è sempre somma tra interi
        - `long` risultato = `(long)` a + b; // Corretto

NOTE

- un letterale intero viene considerato int
- a un long si può assegnare un int (cast implicito)
- si può forzare un letterale inetro a long col suffisso `L`
- Mai fare `==` tra float, solo relazioni (`>`, `<`, ...) o differenza

Letterali in virgola mobile

- Notazione normale o scientifica

```
double d = 126.5;
double d = 1.265E+2;
```

- Per default il tipo è double

```
float f = 3.14; // Errore
```

- Cast esplicito abbreviato: il suffisso `F` (oppure `f`)

```
float f = 3.14F; // OK
float f = (float)3.14;
```

Tipo di dato booleano

- **boolean** : può assumere solo i valori `true` e `false`
    - Non è sicuro usare `0` e `1`

```
boolean b = true;
...
b = false;
```

Tipo di dato carattere

- Utilizza la codifica UNICODE UTF - 16
- Si possono rappresentare la maggior parte dei caratteri degli alfabeti del mondo
- Si può assegnare direttamente o tramite valore Unicode esadecimale:

```
char primoCarattere = 'a';
char car = '@';
char letteraGrecaPhi = '\u03a6';
```

- Si possono usare i caratteri escape come in C:
    - \n, \t, \, ', ", ...

Classi wrapper

- Definite nella libreria standard in corrispondenza dei tipi primitivi:
    - Integer -> int
    - Short -> short
    - Float -> float
    - ...
- Sono interscambiabili con tipi primitivi grazie a autoboxing-autounboxing

Nota: Esistono anche due classi per numeri a precisione arbitraria `BigDecimal` e `BigInteger` definite in `java.math`

Costanti statiche in Float e Double

- Float.NaN
- Float.NEGATIVE_INFINITY
- Float.POSITIVE_INFINITY
- Double.NaN
- Double.NEGATIVE_INFINITY
- Double.POSITIVE_INFINITY

```
double d = -10.0 / 0.0; // 0.0 è approssimato = 0.0000000000000001 per es.
System.out.println(d); // Stampa "-Infinity"
```

- Lo stesso non vale per gli interi ovviamente, 0 è 0.

Underscore in letterali numerici

- Per migliorare la leggibilità si possono usare `_`

```
int i = 1_000_000_000;
```

- Limitazioni:
    - NO ad inizio o fine numero
    - NO adiacente a punto decimale
    - ...