Ripetizioni Francesco

Marini Mattia

2025

Indice

Esercizi

1 Java

Java è molto simile dal punto di vista della sintassi al c++. Non sarà molto complicato il passaggio

1.1 Aspetti simili a c++

La sinttassi di Java è molto simile a quella di c++, ecco gli aspetti che rimangono invariati o quasi:

1.1.1 Dichiarazione delle variabili:

Sinstassi	Differenze rispetto al `c++`
`int x = 15`	invariato
`long x = 15`	invariato
`float x = 15.0f`	nota il `15.0f`, dove `f` sta per float
`double x = 15.0`	float ma con precision maggiore, 64 bit
`boolean x = true`	`boolean` anzichè `bool`
`String s = "stringa"`	dato `String` sarebbe una classe ma è trattato come tipo primitivo, dato che è usato molto frequentemente
`String v[] = new String[15]`	vettore di stringhe di dimensione 15
`int v[] = new int[15]`	vettore di interi di dimensione 15

1.1.2 Commenti:

- Commento riga singola: `// commento`
- Commento righe multiple: `/* commento */`

1.1.3 Cicli:

- Ciclo for:

```
for(int i = 0; i<15; i++){
  // qualcosa
}
```

- Ciclo while:

```
while(i < 15){
  // qualcosa
}
```

1.1.4 Operatori

Operatore	Descrizione
`+` `-` `*` `/`	operatori matematici
`&&`	and logico
`||`	or logico
`!`	not logico

1.1.5 If e switch

```
if(a<b) {
  //codice
}
else(if a>b) {
  //codice
}
else {
  //codice
}
```

```
switch(espressione) {
  case x:
    // codice
    break;
  case y:
    // codice
    break;
  default:
    // codice
}
```

1.1.6 Funzioni

```
void nome_funzione1 (int arg_1, String arg_2){
  //corpo funzione
}

int nome_funzione2 (int arg_1, String arg_2){
  //corpo funzione
  return 5;
}
```

Una funzione è quindi definita indicando nel seguente ordine, esattamente come in c++:

1. Tipo di ritorno (o `void` se non ritorna nulla)
2. Nome della funzione
3. Parametri, racchiusi fra parentesi tonde e separati da virgole
4. Corpo della funzione fra graffe

1.2 Differenze

1.2.1 Stampa su terminale

Una delle feature usata moltissimo, ma completamente diversa dal c++ è la stampa su terminale:

```
System.out.println("Stampa questa cosa");
//stampa andando a capo prima di stampare

System.out.print("Stampa questa cosa");
//stampa SENZA andando a capo
```

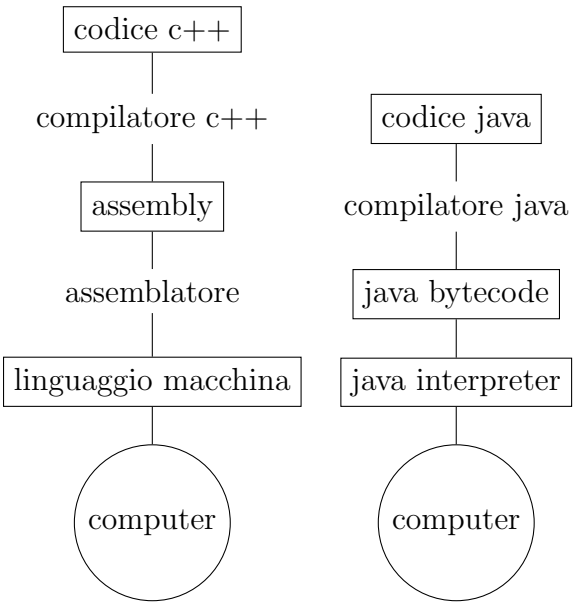
nota che le stringhe si possono concatenare con l'operatore +:

```
String a = "Hello";
String b = "world";

System.out.println(a + b);
//stampa "Hello world"

String c = a + b;
//Inizializza c a "Hello world"
```

1.2.2 Linguaggio interpretato vs compilato

A differenza di c++, java è un linguaggia interpretato

- *Linguaggio compilato*: il codice è "dato in pasto" a un compilatore, il quale lo converte in linguaggio macchina (di fatto in una sequenza di, 0 ed 1)
- *Linguaggio interpretato*: il codice è "dato in pasto" ad un compilatore, il quale lo converte però in bytecode, ossia un linguaggio di basso livello (molto difficile da leggere e scrivere), il quale è in grado di essere letto da un *interprete*



1.2.3 Memoria e puntatori

Java è un linguaggio ad alto livello che gestisce la memoria in maniera diversa rispetto al c++:

- c++: il compito di allocare e deallocare la memoria non più utilizzata è del programmatore
- java: la memoria viene deallocata in maniera automatica tramite un meccanismo chiamato garbage collection

Visto che in java la memoria è gestita in in maniera automatica, il programmatore non ne ha accesso diretto tramite puntatori: al contratio, i puntatori non esistono

1.2.4 Object orientation

Sebbene c++ sia un linguagio che permette di utilizzare classi ed oggetti, in java l'object orientation è forzata: ogni parte del programma deve essere contenuta all'interno di una classe

1.3 Esempio programma

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

Cose da notare:

- La funzione main è contenuta all'interno di una classe "HelloWorld", il cui nome è arbitrario
- La funzione main è marcata come `static`, ciò vuol dire che la funzione esiste anche se non esiste un oggetto di tipo "HelloWorld", affroteremo meglio il modificatore `static` più avanti, per ora possiamo ignorarlo
- La funzione main è marcata come `public`, ciò vuol dire che la funzione è accessibile ovunque. Afffronteremo meglio questo modificatore più avanti, per ora possiamo ignorarlo
- La funzione main prende in ingresso un vettore di stringhe. Nel caso si avviasse l'applicazione da terminale e possibile passare al main dei parametri nel seguente modo:

  ```
  cd cartella_applicazione
  ./nome_applicazione parametro_1 parametro_2 ...
  ```

  in questo caso il vettore di stringhe `args` conterrà `parametro_1` e `parametro_2`. Penso non lo userete mai ma è buono saperlo

1.4 Funzioni utili

In java sono definite alcune funzioni utilissime. Qui una lista (non esaustiva) delle più comuni:

1.4.1 Stringhe

Supponiamo di avere `String s = "stringa stringa";`

Funzione	Descrizione
`s.lenght()`	Ritorna il numero di caratteri conenuti nella stringa (7 nel caso d'esempio)
`s.charAt(int index)`	Ritorna il carattere in posizione `index`
`s.indexOf(char carattere)`	Ritorna l'indice della prima occorrenza di `carattere` in `s`
`s.indexOf(String stringa)`	Ritorna l'indice della prima occorrenza della sotto-stringa `stringa` in `s`

```
String s = "stringa stringa";
s.lenght(); // 15
s.charAt(2); // 'r'
s.indexOf('r'); // 2
s.indexOf("ga"); // 5
```

1.4.2 Vettori

Supponiamo di avere `int v[] = new int[15];`

Funzione	Descrizione
`v.lenght`	ritorna il numero di elemnti contenuti nel vettore

1.5 Parti utili della libreria standard

La libreria standard di java offre moltissime classi utili. Vediamo qui le più comuni

1.5.1 Math

Funzione	Descrizione
`Math.exp(float n)`	Ritorna e^n
`Math.log(float n)`	Ritorna $ln\,(n)$
`Math.abs(float x)`	Ritorna $\|x\|$ (valore assoluto di x)
`Math.sin(float x)`	Ritorna $\sin(x)$
`Math.cos(float x)`	Ritorna $\cos(x)$
`Math.tan(float x)`	Ritorna $\tan(x)$
`Math.asin(float x)`	Ritorna $\arcsin(x)$
`Math.acos(float x)`	Ritorna $\arccos(x)$
`Math.atan(float x)`	Ritorna $\arctan(x)$
`Math.max(float a, float b)`	Ritorna l'elemento maggiore fra a e b
`Math.min(float a, float b)`	Ritorna l'elemento maggiore fra a e b
`Math.floor(float x)`	Arrotonda per difetto x
`Math.ceil(float x)`	Arrotonda per eccesso x
`Math.round(float x)`	Arrotonda x

1.5.2 ArrayList

Utile per avere un vettore di lunghezza variabile. Si inizializza nel seguente modo:

```
ArrayList< tipo > nomeArray = new ArrayList< tipo >(dimensione)
```

Nota che:

- `dimensione` si può omettere, ottenendo un vettore con dimensiane nulla
- `tipo` deve essere una classe. Se voglio un arrayList di tipi primitivi devo utilizzare le classi wrapper (`Integer, Boolean, Double` ...)

Metodo	Descrizione
`v.get( index )`	ritnorna l'elemento all'indice `index`
`v.set( index , element)`	setta l'elemento a indice `index`
`v.add( element )`	inserisce `element` in fondo
`v.remove ( index )`	rimuove l'elemento a indice `index`
`v.size()`	ritorna il numero di elementi conetnuti
`v.clear()`	rimuove tutti gli elementi

Inoltre, sono disponibili features di java per inizializzare arraylist in maniera conveniente:

```
ArrayList<Integer> lista = new ArrayList<>(Arrays.asList(1,2,3));
```

1.5.3 Scanner

La classe scanner ci permette di leggere input utente da terminale (e anche da file, ma non ci servirà). Uno scanner si inizializza così:

```
Scanner nomeScanner = new Scanner(System.in);
```

Per leggere l'input da terminale ci sono i seguenti comandi:

Metodo	Descrizione
nextBoolean( )	Legge un boolean da terminale
nextByte()	Legge un byte da terminale
nextDouble()	Legge un double da terminale
nextFloat ()	Legge un float da terminale
nextInt()	Legge un int da terminale
nextLine()	Legge una String da terminale
nextLong( )	Legge un long da terminale
nextShort()	Legge uno short da terminale

Il metodo più comune è `nextLine()`, dato che ci restituisce l'intera riga come stringa, anche se contiene numeri

1.5.4 Random

Random fornisce un modo comodo per generare numeri casuali. Inizializza con

```
Random nome = new Random()
```

Metodo	Descrizione
nextInt(range)	genera un numero casuale nel range $[0, \text{range})$
nextFloat()	genera un float in range $[0.0, 1.0]$
nextDouble()	genera un double in range $[0.0, 1.0]$

1.5.5 LinkedList

Lista linkata. Utilizzabile in modo molto simile al `ArrayList`. L'accesso agli elementi è molto più lento, rimozioni inserimenti sono molto più veloci

1.5.6 Altre strutture dati utili

- `HashSet`: insieme matematico, possibile vedere se un elemento è contenuto in esso in maniera efficiente
- `Map`: utile poter collegare ad ogni valore un altro valore detto chaive. Si può risalire al valore tramite la chiave in maniera efficiente
- `Stack`
- `Queue`
- `PriorityQueue`: struttura nella quale è possibile accedere all'elemento maggiore in maniera efficiente
- `SortedSet`: struttra nella quale i dati mantengono sempre un ordinameto crescente. Non ammette duplicati

1.6 Lettura/scrittura su file

1.7 Try catch ed eccezioni

In java e in molto linguaggi di programmazione moderni vengono forniti meccanismi standard per gestire situazioni inaspettate che non possono però essere previste con certezza (es. un file non esiste, l'utente ha inserito un dato non valido etc etc). Per queste situazioni esistono oggetti dette *eccezioni*, fatte per descrivere l'errore che si è verificato.

L'idea generale è che qualunque pezzo di codice può sollevare o tirare un'eccezione e, nel caso in cui questa non venga gestita (ossia circondata da un blocco try catch nella porzione di programma chiamante), allora l'applicazione termina, stampando una descrizione della eccezione avvenuta

Il codice che tira eccezioni va circondato da un blocco try catch come segue:

```
try {
    // codice che potrebbe sollevare un'eccezione
} catch (TipoEccezione1 e1) {
    // codice per gestire l'eccezione di tipo TipoEccezione1
} catch (TipoEccezione2 e2) {
    // codice per gestire l'eccezione di tipo TipoEccezione2
} finally {
    // codice che viene eseguito sempre, sia che si sia verificata
   un'eccezione sia che non si sia verificata
}
```

Ad esempio, se avessimo

```
try {
  f1() // solleva eccezione TipoEccezione2
} catch (TipoEccezione1 e1) {
  System.out.println("e1");
} catch (TipoEccezione2 e2) {
  System.out.println("e2");
} finally {
  System.out.println("finally");
}
// stampa:
// e2
// finally
```

Se invece facessimo qualcosa di questo tipo all'interno del main:

```
f1() // solleva eccezione TipoEccezione2
```

allora il programma si interromperebbe stampando informazioni riguardo l'eccezione sollevata

1.7.1 Checked vs unchecked

Esistono due tipi di eccezioni: *checked* e *unchecked*. Dal punto di vista logico le prime rappresentato tutte le condizioni che non NON abbiamo modo di verificare non accadano, mentre nel secondo, utilizzando determinate accortezze possiamo evitare succedano. Per questa ragione, siamo *obbligati* a gestire le prime, mentre possiamo ignorare le seconde.

Checked exceptions	Unchecked exceptions
IOException	ArithmeticException
ClassNotFoundException	NullPointerException
FileNotFoundException	ArrayIndexOutOfBoundsException
SQLException	IllegalArgumentException

L'idea è che le unchecked exceptions possono essere ignorate proprio in virtù del fatto che tramite determinate accortezze possiamo evitare che si verifichino (es. controllare che un indice sia valido prima di accedere ad un array, controllare che un divisore non sia 0 prima di effettuare una divisione etc etc), mentre ciò non è vero per quanto riguarda le checked

Ciò che cambia nel pratico è che quando utilizziamo una porzione di codice che può sollevare una checked exception, il compilatore ci obbliga a circondarla con un blocco try catch, oppure a dichiarare che la funzione che la contiene può sollevare quell'eccezione (usando la parola chiave `throws`):

```
public f1() throws CheckedException{}
```

Si noti che in realtà è possibile dichiarare una funzione con la keyword throw anche per le unchecked exceptions, ma non è obbligatorio

1.8 Scanner

Scanner fornisce il modo più semplice (ma più lento) per leggere da file o da `stdin`:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

try (Scanner scanner = new Scanner(new File("path/to/file.txt"))) {
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        // Process the line
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

per la scrittura invece il metodo preferito è il seguente:

```
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.IOException;

try (PrintWriter writer = new PrintWriter(new FileWriter("esempio.txt",
    true))) {
    writer.println("Riga aggiunta!");
} catch (IOException e) {
    e.printStackTrace();
}
```

1.9 File

Il package nio.file un modo più rapido e moderno per leggere da file:

```
package com.mycompany.app;

import java.nio.file.Files;
import java.nio.file.Paths;
import java.io.IOException;

String content=new
    String(Files.readAllBytes(Paths.get("path/to/file.txt")));
List<String>lines=Files.readAllLines(Paths.get("path/to/file.txt"));
```

per la scrittura invece il metodo preferito è il seguente:

```
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.io.IOException;

public class AppendFileNio {
    public static void main(String[] args) {
        String nuovaRiga = "Nuova riga aggiunta!";
        try {
            Files.write(
                Paths.get("esempio.txt"),
                Arrays.asList(nuovaRiga),
                StandardOpenOption.APPEND
            );
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

1.10 Esercizi

Collezione di esercizi, divisi per categoria

Esercizio 1: *Hello World (hello_world)*

Scrivere un programma in Java che prenda in input una stringa `s` e un numero `n` e stampi `s` `n` volte, accompagnata dal numero di riga. Ad esempio, dati in input `userin` e `4`, l'otput sarà:

1. `userin`
2. `userin`
3. `userin`
4. `userin`

Esercizio 2: *Hello classe (estrazione)*

Crea un programma per giocare a una sorta di roulette modificata. Le regole sono le seguenti:

- Vengono estratte delle palline che hanno un colore (`rosso`, `giallo` o `verde`) e un numero da 1 a 9 estremi compresi
- Il giocatore deve indicare un numero e un colore
- Il punteggio viene assegnato così:
  - Colore giusto: `+1`
  - Un punteggio che varia da 0 a 4 in base alla distanza del numero previsto da quello estratto, ad esempio, detto n_e il numero *estratto* e n_p il numero *previsto*
    $$\left(1 - \frac{|n_e - n_p|}{8}\right) \cdot 4$$
  - Se colore e numero sono uguali: `+1`

Il programma deve chiedere in input all'utente un numero e un colore ed estrarre una pallina casualmente, per poi stampare in ouput il punteggio ottenuto. Si usi una classe `ball` e una classe `game` per gestire le partite.

Il programma deve partire chiedendo all'utente il numero n di round che vuole giocare. Dopodiché, verranno chieste n previsione all'utente; per ognuna deve essere stampato il punteggio e alla fine degli n round deve essere visualizzato il punteggio totale

Esercizio 3: *Geometria 1 (geometry1)*

Creare una classe per ciascuno dei seguenti oggetti geometrici: `Punto`, `Rettangolo`, `Cerchio`. Creare una classe `PianoCartesiano` che possa contenere le tre classi elencate precedentemente. Una volta inserita una classe nel piano cartesiano verrà ritornato un `id`, tramite il quale possiamo accedervi (l'istanza dell'oggetto aggiunto rimane scollegata da quella inserità nel piano cartesiano). Creare un metodo `closer` che prenda un `id` esistente e ritorni l'id della forma geometrica più vicina (usare il centro delle figure per confrontare la distanza)

Esercizio 4: *Geometria 1 (geometry1)*

Creare una classe per ciascuno dei seguenti oggetti geometrici: `Punto`, `Rettangolo`, `Cerchio`. Creare una classe `PianoCartesiano` che possa contenere le tre classi elencate precedentemente. Una volta inserita una classe nel piano cartesiano verrà ritornato un `id`, tramite il quale possiamo accedervi (l'istanza dell'oggetto aggiunto rimane scollegata da quella inserità nel piano cartesiano). Creare un metodo `closer` che prenda un `id` esistente e ritorni l'id della forma geometrica più vicina (usare il centro delle figure per confrontare la distanza)

Esercizio 5: *Montecarlo (montecarlo)*

Il casinò di Montecarlo ha richiesto una nuova slot machine; lo scopo del gioco è quello di simulare la pesca di *cinque carte* da una mazzo standard (con quattro semi, le carte da 2 a 10, l'asso(1) le tre figure (11,12,13) e niente jolly)

Le opportunità di vincita (con quattro delle 5 carte estratte) sono le seguenti:

Combinazione	vincita
Poker di figure	1000 €
Poker d'assi	2000 €
[a] Colore	Somma delle carte uscite × 8
[b] Scala colore	Somma delle carte uscite × 10

La presenza di un'altra carta non intacca la vincita.

Contare quante slotmachine sono presenti nel casino

Si scrivano le classi opportune per implementare il gioco e un classe `Casino` dove create diverse `Slot`.

Fare in modo che la giocata possa essere efffetutata solo dopo aver inseirto un certo importo;

Introdurre successivamente anche i *jolly* tra le carte e fare in modo che la presenza del jolly come carta estratta raddoppi la vincita;

[a] 5 carte dello stesso seme, ad es: 1-4-5-10-13, *tutti cuori* → vincita = 264 €
[b] 5 carte dello stesso seme in sequenza, ad esempio: 3-4-5-6-7 *tutte picche* → vincita = 250 €

Esercizio 6: *Sistema gestione prenotazioni centro sportivo*

Il Centro Sportivo *ActiveWorld* offre diverse attività (*nuoto*, *palestra*, *yoga*, *arrampicata*) e vuole un software che gestisca le prenotazioni giornaliere per i vari corsi e spazi. Tutte le informazioni devono essere salvate e gestite tramite file di testo, letti all'avvio dell'applicazione e aggiornati durante l'esecuzione.

File coinvolti: l'applicazione utilizza tre file:

- *Utenti registrati:* ogni utente è caratterizzato da:
  - nome
  - cognome
  - data di nascita
  - telefono
  - email
- *Attività disponibili*
  - nome dell'attività

- capacità massima
- fascia oraria (es. `18:00--19:30`)

- *Prenotazioni attive:* ogni riga del file contiene:
  - email dell'utente
  - nome attività
  - data della prenotazione (`gg/mm/aaaa`)

Funzionalità richieste

1. *Caricamento dati dai 3 file:* all'avvio, il programma deve:
   - leggere tutti gli utenti registrati
   - leggere tutte le attività disponibili
   - leggere le prenotazioni attive
   - verificare che ogni prenotazione faccia riferimento a un utente e a un'attività validi
   - segnalare eventuali incongruenze (es. utente inesistente → prenotazione ignorata)
2. *Visualizzare le attività disponibili:* mostrare
   - nome attività
   - fascia oraria
   - numero prenotazioni attuali
   - posti disponibili rimanenti
3. *Visualizzare tutte le prenotazioni:* raggruppate per attività, ordinate per data.
4. *Cercare le prenotazioni di un utente:* richiedere *email* e mostrare tutte le sue prenotazioni.
5. *Aggiungere una nuova prenotazione:* il programma deve:
   - chiedere: email utente, attività, data
   - verificare che l'utente esista
   - verificare che l'attività esista
   - controllare che la data non sia nel passato
   - controllare che l'utente non abbia già una prenotazione per la stessa attività nella stessa giornata
   - controllare che l'attività non sia al completo per quella data
   - se tutto ok:
     - aggiungere la prenotazione in memoria

- aggiungerla al file in modalità *append*

6. *Cancellare una prenotazione*
   Richiedere:
   - email utente
   - nome attività
   - data

   Controllare che esista, poi:
   - rimuoverla dall'elenco
   - aggiornare il file riscrivendolo completamente
7. *Report automatico delle attività quasi piene:* il programma deve offrire una funzione *Genera Report* che:
   - mostra tutte le attività che hanno meno del 10% dei posti liberi nella data selezionata
   - genera anche una versione salvata in un file `report.txt` con formattazione ordinata
8. *Statistiche settimanali (opzionale):* dato un range di date, il programma deve indicare:
   - quale attività è stata prenotata di più
   - quale di meno
   - il numero totale di prenotazioni
   - percentuale di riempimento media delle attività

Requisiti e vincoli tecnici

- Il formato dei file deve rimanere sempre uniforme.
- Se vengono inseriti dati mancanti o non validi, il programma deve richiederli di nuovo.
- Ogni modifica deve aggiornare coerentemente i file.
- L'applicazione deve funzionare finché il gestore non sceglie di uscire.

Esercizio 7: *ASCII art*

Creare una applicazione che, dato in input un file *in formato* `.ppm p3` crei una ascii art di quanto rappresentato dalla figura. Il main deve definire due parametri:

- `inputFile` il file `.ppm` da convertire in ascii art
- `outputFile` il file `.txt` di output contenente la ascii art

Il formato `.ppm p3` rappresenta una immagine tramite una matrice di pixel, ognuno dei quali è rappresentato da tre numeri interi (rosso, verde, blu) che vanno da 0 a 255. Il file inizia con una intestazione composta da:

```
P3
<WIDTH> <HEIGHT>
<MAX_COLOR>
<R> <G> <B> ... <R> <G> <B>
```

Ad esempio, un file contenente 3 pixel in orizzontale, in ordine, uno rosso, uno verde e uno blu, sareebbe così codificato:

```
P3
3 1
255
255 0 0 0 255 0 0 0 255
```

dove

$$\underbrace{255\ 0\ 0}_{\text{pixel 1}}\ \underbrace{0\ 255\ 0}_{\text{pixel 2}}\ \underbrace{0\ 0\ 255}_{\text{pixel 3}}$$

hint: si può usare la seguente formula per convertire un pixel in bianco e nero:

```
double grayscale = 0.299 * r + 0.587 * g + 0.114 * b;
```

Esercizio 8: *Operazioni arraylist*

Scrivere un programma che chieda all'utente un numero `n` di interi da terminale. L'applicazione deve innanzitutto chiedere `n` all'utente e successivamente leggere `n` interi in un arraylist. Una volta letti tutti i numeri viene stampato un prompt che chiede quale operazione eseguire. Le opzioni sono le seguenti:

1. *Verifica numero*: viene chiesto un numero e l'applicazione deve verificare se il numero è presente nell'arraylist
2. *Rimuovi numero*: viene chiesto un numero e l'applicazione deve rimuovere *tutte* le occorrenze di quel numero nell'arraylist
3. *Sort*: chiede se vuole ordinare l'arraylist in ordine crescente o decrescente e stampa il risultato ordinato. Usare un algoritmo di sort scritto a mano, non quello implementato nelle classi standard!
4. *Check sort*: stampa `true` se il vettore è ordinato, `false` altrimenti
5. *Reverse*: inverte l'ordine del vettore e lo stampa
6. *Quit*: esce dall'applicazione