Caratteri – Input da Console

Il tipo char

Input dallo standard input

Il tipo base char

- Il tipo base **`char`** rappresenta i caratteri
- Come sappiamo Java gestisce tutto il set di caratteri Unicode
- Per indicare un carattere basta inserirlo tra due apici singoli: ‘a’
- Ogni sequenza di escape corrisponde a un carattere ‘\n’, ‘\t’, ‘\u009F’
- Un carattere ha associato un valore numerico reperibile con Character.getNumericValue

Caratteri e Stringhe

- ‘h’ è un char
- “h” è una stringa, quindi un oggetto dello Heap, che contiene un solo carattere ‘h’
- “Pippo”.charAt(0) ritorna il carattere ‘P’
- Nelle stringhe i caratteri sono numerati da 0 in su
- “Pippo”.length() ritorna 5, la lunghezza della stringa
- Consultare le API

Input da console

- Oltre che da una finestra di dialogo l'input può essere prelevato da uno stream di ingresso
- In Java, così come in C e in altri linguaggi, esiste uno stream apposito per questo che viene chiamato **standard input**
- Conosciamo già **`System.out`**, che è lo standard output, e sappiamo che è un oggetto della classe **`java.io.PrintStream`**
- **`System.in`** è un oggetto della classe **`java.io.InputStream`** ed è lo standard input

Ottenere l'input

- Un oggetto della classe **`InputStream`**, quale è **`System.in`**, è in grado di leggere un **byte** per volta dallo stream che rappresenta
- Non è molto comodo
- Quello che vorremmo è poter ottenere, come con la finestra di dialogo, una stringa di input
- Per prima cosa bisogna incapsulare l'oggetto **`System.in`**, di tipo **`InputStream`**, in un oggetto della classe **`java.io.InputStreamReader`**

Ottenere l'input

- Un oggetto della classe **`InputStreamReader`** interpreta i byte di un oggetto InputStream come caratteri (tipo base char di Java)
- È un passo avanti per arrivare al nostro obiettivo
- I costruttori di questi oggetti richiedono sempre come argomento un'oggetto della classe **`InputStream`**
- Possiamo quindi usare **`System.in`**

Ottenere l'input

```
InputStreamReader reader = new
  InputStreamReader(System.in);
```

- **`reader`** può restituire l'input sotto forma di un carattere per volta (guardare le API e scoprire come mai il metodo **`read`** restituisce un **`int`** piuttosto che un **`char`**)
- Possiamo ottenere di meglio
- Gli oggetti della classe **`java.io.BufferedReader`** possono restituire stringhe formate da caratteri di uno stream di caratteri

Ottenere l'input

```
BufferedReader console = new
  BufferedReader(reader);
```

- Il costruttore richiede un oggetto della classe **`java.io.Reader`**, di cui **`InputStreamReader`** è una sottoclasse
- Possiamo quindi passare il nostro oggetto **`System.in`**, incapsulato nell'oggetto **`reader`** di tipo **`InputStreamReader`**, al costruttore e ottenere l'oggetto riferito da **`console`**

Ottenere l'input

- Sugli oggetti della classe **`BufferedReader`** è possibile chiamare il metodo **`readLine()`** che restituisce una stringa contentente una **linea** di testo
- Questo è quello che volevamo ottenere

Ottenere l'input

- Ricapitolando:

```
InputStreamReader reader = new
  InputStreamReader(System.in);
BufferedReader console = new
  BufferedReader(reader);
```

- Oppure

```
BufferedReader console = new
  BufferedReader(new
  InputStreamReader(System.in));
```

Ottenere l'input

- A questo punto:

```
String input = console.readLine();
```

- Aspetta fino a quando l'utente non digita una linea di testo e preme Invio
- La linea scritta è contenuta nella stringa riferita da **`input`**
- A questo punto possiamo fare il parsing della stringa con i metodi delle classi involucro se ci aspettiamo l'inserimento di un valore di un certo tipo (es **`Integer.parseInt`**)

IOException

- Abbiamo già visto che il metodo **`Integer.parseInt`** può sollevare un'eccezione se la stringa passata non contiene le cifre di un intero
- Anche il metodo **`readLine`** della classe **`BufferedReader`** può sollevare un'eccezione se qualcosa va storto con il reperimento dell'input
- L'eccezione è del tipo **`IOException`**
- L'eccezione **`IOException`** è un'eccezione che deve essere gestita obbligatoriamente
- Nel caso non la si voglia gestire si deve esplicitare che il metodo che stiamo scrivendo la può sollevare

Throws

- Nel nostro caso il metodo che chiama **`readLine`** è il main
- Siccome ancora non abbiamo visto come gestire l'eccezione dobbiamo esplicitamente dichiarare che non la gestiamo scrivendo

```
public static void main(String[]
        argv) throws IOException {
...
}
```

Un programma di esempio

```
import java.io.*;
public class InputDaConsole  {
  public static void main(String argv[]) throws
                                        IOException {
    BufferedReader console = new
          BufferedReader(new
               InputStreamReader(System.in));
    System.out.println("Quanti nickels hai?");
    String input = console.readLine();
    int nickels = Integer.parseInt(input);
    System.out.println("Hai scritto " + nickels);
  }
}
```

Esercizio

- Riscrivere il costruttore della classe Purse in modo che chieda in input il numero di monete iniziali
- Aggiungere alla classe Purse un metodo che restituisca il totale in dollari e in penny (100 penny = 1 dollaro) (Suggerimento: usare la divisione intera e il resto della divisione intera)

Esercizio

- Scrivere una classe RisolutoreEquazione2Grado
- Il costruttore deve chiedere in input i coefficienti a, b, c dell'equazione
- Implementare i due metodi per dare le due soluzioni:
  - public double getFirstSolution()
  - public double getSecondSolution()
- Scrivere una classe test

Esercizio

- Scrivere un programma che assista un cassiere nel dare il resto
- Input: sommaDaPagare e sommaRicevuta
- La differenza sommaRicevuta – sommaDaPagare rappresenta il resto da dare
- Il resto deve essere corrisposto usando le seguenti banconote/monete: 1 dollaro, 1 Quarter, 1 Dimes, 1 Nickel, 1 Penny
- Il programma deve indicare quante monete di ogni tipo il cassiere deve dare --continua→

Esercizio cont'd

- Il programma deve fornire una soluzione che corrisponda all'erogazione del minimo numero possibile di banconote/monete
- Esempio

```
Cassiere harry = new Cassiere();
harry.setSommaDaPagare(8.37);
harry.riceve(10);
int dollars = harry.returnDollars(); // Restituisce 1
int quarters = harry.returnQuarters(); // Restituisce 2
int dimes = harry.returnDimes(); // Restituisce 1
int nickels = harry.returnNickels(); // Restituisce 0
int pennies = harry.returnPennies(); // Restituisce 3
```

Esercizio

- Scrivere un programma che legge un numero intero e poi stampa le sue cifre, una ad una, in ordine inverso
- Esempio: se leggo 78349
- Deve stampare
- 9
- 4
- 3
- 8
- 7

Esercizio

- Scrivere un programma che calcoli la data della domenica di Pasqua, che è la prima domenica dopo la prima luna piena di primavera, di un qualunque anno.
- Si usi il seguente algoritmo ideato da Carl Friedrich Gauss nel 1800:
- Sia **`y`** l'anno (1800, 2001, ...)
- Dividi **`y`** per 19 ottenendo il resto **`a`**. Ignora il quoziente
- Dividi **`y`** per 100 ottenendo il quoziente **`b`** e il resto **`c`**
- Continua →

Esercizio cont'd

- Dividi **`b`** per 4 ottenendo quoziente **`d`** e resto **`e`**
- Dividi 8 * **`b`** + 13 per 25 ottenendo il quoziente **`g`**. Ignora il resto.
- Dividi 19 * **`a`** + **`b`** - **`d`** - **`g`** + 15 per 30 ottenendo il resto **`h`**. Ignora il quoziente.
- Dividi **`c`** per 4 , ottenendo il quoziente **`j`** e il resto **`k`**
- Dividi **`a`** + 11 * **`h`** per 319, ottenendo il quoziente **`m`**. Ignora il resto
- Continua →

Esercizio cont'd

- Dividi 2 * **`e`** + 2 * **`j`** - **`k`** - **`h`** + **`m`** + 32 per 7 ottenendo il resto **`r`**. Ignora il quoziente
- Dividi **`h`** - **`m`** + **`r`** + 90 per 25, ottenendo il quoziente **`n`**. Ignora il resto
- Dividi **`h`** - **`m`** + **`r`** + **`n`** + 19 per 32 ottenendo il resto **`p`**. Ignora il quoziente
- **Pasqua cade il giorno p del mese n**
- Scrivere una classe **`Year`** con metodi **`getEasterDay()`** e **`getEasterMonth()`**