Linguaggi di Programmazione

a.a. 13/14

docente: Gabriele Fici

gabriele.fici@unipa.it

9 - Input/Output

9 - Input/Output

- La classe `File`, presente nella libreria `java.io`, permette di gestire file e cartelle presenti nel file system
- Si può dunque mettere un file in un riferimento di tipo `File`

Esempio:

```
import java.io.* ;

File f = new File(“input.txt”);
// f è ora un riferimento al file input.txt
```

9 - Input/Output

- Per sapere se un file esiste, si usa il metodo `exists`
- Per leggere il contenuto di un file di testo, si può usare uno Scanner

Esempio:

```
import java.io.* ;
import java.util.* ;

File f = new File(“input.txt”);
if ( f.exists() ) {
   Scanner in = new Scanner (f);
}
```

9 - Input/Output

- Possiamo a questo punto leggere il file, ad es. riga per riga, usando i metodi `nextLine` e `hasNextLine`:

```
import java.io.* ;
import java.util.* ;

String s;
Scanner in = new Scanner(new File("input.txt"));
   while (in.hasNextLine()){
        s = in.nextLine();
        ...
   }
```

9 - Input/Output

- ...oppure possiamo leggere il file parola per parola, usando i metodi `next` e `hasNext`:

```
import java.io.* ;
import java.util.* ;

String s;
Scanner in = new Scanner(new File("input.txt"));
   while (in.hasNext()){
        s = in.next();
        ...
   }
```

9 - Input/Output

- Per usare i file bisogna aggiungere al metodo `main` la seguente dicitura (vedremo poi perché):

`throws IOException`

Esempio:

```
public static void main(String[] args)throws IOException{

 String s;
 Scanner in = new Scanner(new File("input.txt"));
    while (in.hasNext()){
        s = in.next();
        System.out.print(s.trim() + " ");
    }
 in.close();
} //cosa fa questo programma?
```

9 - Input/Output

- E' importante chiudere l'oggetto Scanner col metodo `close` prima di terminare il programma, per evitare la mancata scrittura dei dati

```
public static void main (String[] args) throws IOException{

 String s;
 Scanner in = new Scanner(new File("input.txt"));
    while (in.hasNext()){
        s = in.next();
        System.out.print(s.trim() + " ");
    }
 in.close();
}
```

9 - Input/Output

- Per scrivere su un file di testo, si può usare la classe `PrintWriter`, presente in `java.io.Writer`
- Si costruisce un oggetto di tipo `PrintWriter` fornendo il nome del file su cui si vuole scrivere
- Se il file non esiste viene creato, se esiste viene sovrascritto

Esempio:

```
import java.io.* ;

PrintWriter out = new PrintWriter ("output.txt");
```

9 - Input/Output

- Sull'oggetto di classe PrintWriter si possono invocare i noti metodi `print`, `println` e `printf`

Esempio:

```
import java.io.* ;
import java.util.* ;

String testo = "ciao!";

PrintWriter out = new PrintWriter ("output.txt");
out.printf("*\t%-20s*\n", testo);
out.close();
```

9 - Input/Output

- Anche qui, ricordiamoci di chiudere il PrintWriter col metodo `close` prima di terminare il programma!

```
import java.io.* ;
import java.util.* ;

String s = “ciao!”;

PrintWriter out = new PrintWriter (“output.txt”);
out.printf(“*\t%-20s*\n”);
out.close();
```

9 - Input/Output

- Il limite principale della classe `PrintWriter` è che non permette di scrivere su un file in modalità append
- Per ovviare a questo problema, si "filtra" il file con un oggetto della classe `FileWriter` (presente in `java.io.Writer`) la quale ha un costruttore che permette di settare la modalità append (true o false)

```
import java.io.* ;
import java.util.* ;

PrintWriter out =
 new PrintWriter(new FileWriter("output.txt", true));
```

9 - Input/Output

- In Java è anche possibile leggere e scrivere oggetti
- Per questo si usano:
  - la classe `ObjectInputStream` per gestire un flusso di oggetti in lettura
  - la classe `ObjectOutputStream` per gestire un flusso di oggetti in scrittura

9 - Input/Output

- Per potere memorizzare un oggetto, lo si deve trasformare in un flusso di byte
- Questa operazione si chiama *serializzazione*
- Ogni oggetto serializzato riceve un numero di serie nel flusso
- Simmetricamente, la lettura di oggetti da un flusso comporta la deserializzazione degli oggetti

9 - Input/Output

- Una classe, per potere serializzare oggetti, deve implementare l'interfaccia `Serializable`
- Tale interfaccia non ha metodi (serve solo come marcatore, quindi basta dichiararne l'implementazione)
- Ogni classe serializzabile ha un identificativo universale di serializzazione (quindi attenzione: modificando il codice del programma e ricompilandolo non è più possibile usare oggetti serializzati in precedenza!)

9 - Input/Output

- Per creare uno stream di output di oggetti serializzati si usa la classe `ObjectOutputStream`, il cui costruttore ha bisogno di un oggetto di classe `FileOutputStream` (entrambe le classi sono nel segmento `java.io`)
- Per scrivere un oggetto si usa il metodo `writeObject`

Esempio:

```
ContoBancario cb = new ContoBancario();

ObjectOutputStream out =
    new ObjectOutputStream
        (new FileOutputStream ("conti2013.dat"));

out.writeObject(cb);
```

9 - Input/Output

- Per leggere oggetti da uno stream di oggetti in entrata la procedura è analoga, basta sostituire “Out” con “In”
- Per leggere un oggetto si usa il metodo `readObject` (bisogna però fare un cast esplicito per deserializzare)

Esempio:

```
ObjectInputStream in =
    new ObjectInputStream
        (new FileInputStream (“conti2013.dat”));

ContoBancario cb = (ContoBancario) in.readObject();
```

9 - Input/Output

- Altri remark:
  - Tutte le classi coinvolte nella serializzazione devono implementare l'interfaccia `Serializable`
  - I flussi `ObjectInputStream` e `ObjectOutputStream` vanno chiusi (col metodo `close`) quando non servono più
  - al main va aggiunta la dicitura

    `throws IOException, ClassNotFoundException`

9 - Input/Output

Esempio:

```
import java.util.*;
import java.io.*;

class ContoBancario implements Serializable {

  private double saldo;
  private String nomeTitolare;

  public ContoBancario (String titolare, double saldo){
     this.nomeTitolare = titolare;
     this.saldo = saldo;
  }

...
```

9 - Input/Output

```
...

public void deposita (double valuta){
   this.saldo += valuta;
}

public void preleva (double valuta){
   this.saldo -= valuta;
}

public double getSaldo (){
   return this.saldo;
}

public String getTitolare (){
   return this.nomeTitolare;
}

...
```

9 - Input/Output

```
...

  public void effettuaBonifico
              (double ammontare, ContoBancario that){
     this.preleva(ammontare);
     that.deposita(ammontare);
 }

   public String toString() {
       return
       "\nConto corrente: "  +
       "\n\tNome Titolare: " + this.nomeTitolare +
       "\n\tSaldo: " + this.saldo +
       "\n";
   }

}
```

9 - Input/Output

```
public class Banca implements Serializable {

   ArrayList<ContoBancario> conti =
       new ArrayList<ContoBancario>();

   public static void main(String[] args)
       throws IOException, ClassNotFoundException {

   Banca test;

   File f = new File("banca.dat");
   if (f.exists() && f.length() > 0) {
      ObjectInputStream in_stream =
          new ObjectInputStream(new FileInputStream(f));
      test = (Banca) in_stream.readObject();
      in_stream.close();
   }
   else test = new Banca () ;
...
```

9 - Input/Output

```
...

   test.conti.add(new ContoBancario("io", 1500)) ;
   test.conti.add(new ContoBancario("Mamma", 3500));
   test.conti.get(1).effettuaBonifico
                (1500, test.conti.get(0));

   for ( int i = 0 ; i < test.conti.size() ; i++ )
       System.out.println ( test.conti.get(i) );

   ObjectOutputStream out =
       new ObjectOutputStream (new FileOutputStream(f));
   out.writeObject(test);
   out.close();
   }
}
```