

# Programmazione in Java (9)

Mauro Lorenzutti

## Scaletta

- I/O Evoluto
- Serializzazione
- Comunicazioni via socket
- JUnit

## I/O

- Progettare e implementare una gerarchia di classi per astrarre il device di I/O per la classe Persona
- Input da:
  - Tastiera
  - File
- Output su
  - Video
  - File

## Serializzazione

- Perché
- Concetti base
- Un po' di codice
- Esercizio

## Persistenza degli oggetti

- Quando un programma termina tutti gli oggetti muoiono con lui
- Serve un meccanismo per salvare gli oggetti per farli "vivere" indipendentemente dal programma che li ha generati
- Questo fenomeno è detto **persistenza** degli oggetti

## Serializzazione

- Per ottenere la persistenza degli oggetti si può codificarli in un file e poi leggerli → molto scomodo
- Si può serializzarli e scriverli in un file o inviarli attraverso la rete
- Serializzare un oggetto vuol dire rappresentarlo come una sequenza di byte
- Questa rappresentazione in byte può essere usata per ricostruire l'oggetto

## Serializzare su file

```
FileOutputStream fos = new  
    FileOutputStream(nomefile);  
  
ObjectOutputStream oos = new  
    ObjectOutputStream(fos);  
  
oos.writeObject(oggetto);
```

## De-serializzare da file

- Il processo inverso: da un flusso di byte ricostruire l'oggetto originale

```
FileInputStream fis = new  
    FileInputStream(nomefile);  
  
ObjectInputStream ois = new  
    ObjectInputStream(fis);  
  
TipoOggetto oggetto =  
    (TipoOggetto)ois.readObject();
```

## Attenzione!

- L'oggetto da serializzare e tutti i suoi attributi devono essere predisposti alla serializzazione

```
public class Persona implements  
    Serializable{  
  
    private static final long serialVersionUID =  
        1L;
```

## Escludere dal flusso

- È possibile in certi casi avere la necessità di escludere una variabile dal flusso di byte

```
private transient String password;
```

## Esercizio

- Estendere la gerarchia di I/O per includere una classe che permetta il salvataggio su file di dati serializzati e la successiva lettura e ricostruzione di questi

## Comunicazione via socket

- Introduzione
- Socket
- ServerSocket
- Esercizio

## Comunicazioni in rete

- Far comunicare due programmi distinti attraverso una connessione di rete
- Socket
  - È un ponte per comunicare
  - È associata a un host e una porta

## Socket 1/4

- Java mette a disposizione la classe *Socket* (socket client)
- per aprire un canale è necessario specificare host e porta

```
Socket s = new Socket("localhost", 1555);
```
- Attenzione: le porte fra 0 e 1023 sono riservate

## Socket 2/4

- È necessario specificare un timeout entro il quale i dati possono essere letti

```
s.setSoTimeout(30000);
```
- Allo scadere viene generata un'eccezione *InterruptedIOException*

## Socket 3/4

- Ora è possibile usare la socket per inviare e ricevere dati

```
PrintStream out = new PrintStream(s.getOutputStream());  
out.print("Stringa da inviare");
```

```
BufferedReader r = new BufferedReader(new  
    InputStreamReader(s.getInputStream()));  
boolean eof = false;  
while(!eof) {  
    String riga = r.readLine();  
    if (riga!=null)  
        System.out.println(riga);  
    else  
        eof = true;  
}
```

## Socket 4/4

- Al termine è buona regola chiudere la connessione

```
s.close();
```

## ServerSocket 1/2

- Sono i socket lato server
- Un socket lato server controlla una porta TCP e avverte i tentativi di connessione da parte dei client
- Java mette a disposizione la classe *ServerSocket*

## ServerSocket 2/2

- Le connessioni da parte dei client vengono accettate attraverso il metodo *accept()*
- Questo metodo restituisce un *Socket* per gestire la comunicazione

```
ServerSocket server = new ServerSocket(1555);  
Server client = server.accept();
```

## Socket e serializzazione

- Anche in questo caso la serializzazione è molto comoda per permettere a due applicazioni lo scambio di oggetti via socket

```
ObjectOutputStream oos = new  
ObjectOutputStream(client.getOutputStream());  
oos.writeObject(p);
```

```
ObjectInputStream ois = new  
ObjectInputStream(connection.getInputStream());  
Persona p = (Persona)ois.readObject();
```

## Esercizio

- Estendere la classe IOEvoluto per gestire la comunicazione via socket degli oggetti di tipo persona