Metodologie di Programmazione

Lezione 24: Input e output

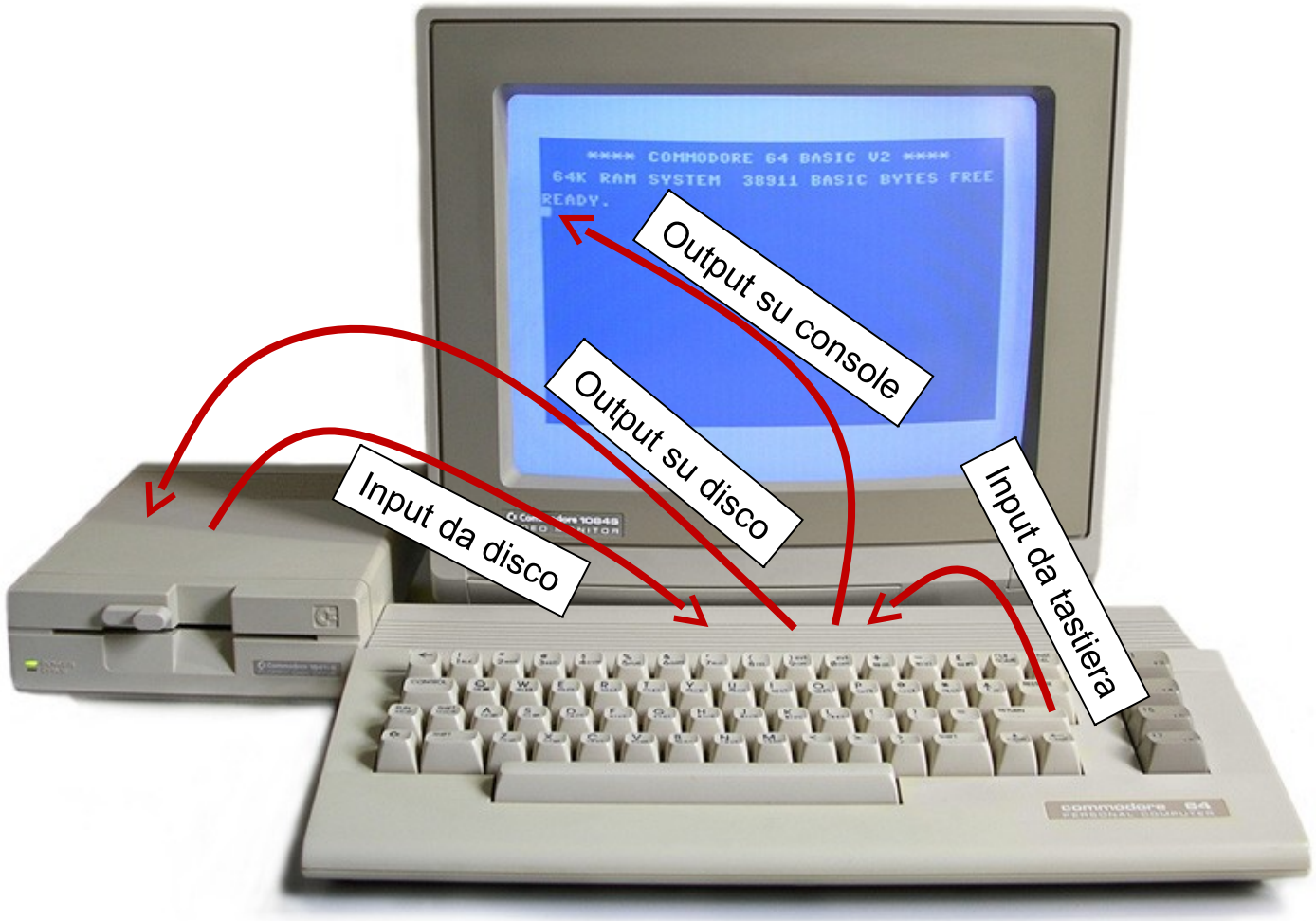
Lezione 24: Sommario

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Input e output in Java
- Serializzazione e deserializzazione di oggetti

Input & Output



Output su console

- Usando la classe System.out (oggetto dello standard output)
- E' un campo statico, pubblico, final di System di tipo java.io.PrintStream (estende java.io.OutputStream)

```
String s = "hello";
System.out.print(s);
System.out.println();
System.out.println(10000000000L);
System.out.print(42.0);
System.out.format("La stringa contiene %d caratteri; il primo carattere è %c\n", s.length(), s.charAt(0));
```

java.io.PrintStream

Method Summary

PrintStream	**append**(char c) Appends the specified character to this output stream.
PrintStream	**append**(CharSequence csq) Appends the specified character sequence to this output stream.
PrintStream	**append**(CharSequence csq, int start, int end) Appends a subsequence of the specified character sequence to this output stream.
boolean	**checkError**() Flushes the stream and checks its error state.
protected void	**clearError**() Clears the internal error state of this stream.
void	**close**() Closes the stream.
void	**flush**() Flushes the stream.
PrintStream	**format**(Locale l, String format, Object... args) Writes a formatted string to this output stream using the specified format string and arguments.
PrintStream	**format**(String format, Object... args) Writes a formatted string to this output stream using the specified format string and arguments.
void	**print**(boolean b) Prints a boolean value.
void	**print**(char c) Prints a character.
void	**print**(char[] s) Prints an array of characters.
void	**print**(double d) Prints a double-precision floating-point number.
void	**print**(float f) Prints a floating-point number.
void	**print**(int i) Prints an integer.
void	**print**(long l) Prints a long integer.
void	**print**(Object obj)

Input da tastiera

- Usando la classe java.util.Scanner costruita con la standard input stream:
- System.in è un campo statico, pubblico, final di System di tipo java.io.InputStream

```
Scanner s = new Scanner(System.in);
int k = s.nextInt();
System.out.println("Hai digitato: "+k);
String string = s.next();
System.out.println("Hai digitato: "+string);
```

java.io.InputStream

All Implemented Interfaces:

Closeable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

---

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

See Also:

BufferedInputStream, ByteArrayInputStream, DataInputStream, FilterInputStream, read(), OutputStream, PushbackInputStream

---

Constructor Summary

InputStream()

Method Summary

int	available() Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
void	close() Closes this input stream and releases any system resources associated with the stream.
void	mark(int readlimit) Marks the current position in this input stream.
boolean	markSupported() Tests if this input stream supports the `mark` and `reset` methods.
abstract int	read() Reads the next byte of data from the input stream.
int	read(byte[] b) Reads some number of bytes from the input stream and stores them into the buffer array `b`.

La classe java.util.Scanner (1)

- Usando la classe java.util.Scanner costruita con la standard input stream:

Constructor Summary
`Scanner(File source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(File source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(InputStream source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(Readable source)` Constructs a new `Scanner` that produces values scanned from the specified source.
`Scanner(ReadableByteChannel source)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(ReadableByteChannel source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(String source)` Constructs a new `Scanner` that produces values scanned from the specified string.

La classe java.util.Scanner (2)

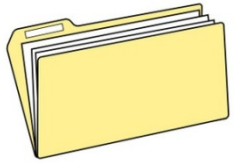
- Alcuni metodi della classe:

Method Summary	
void	close() Closes this scanner.
Pattern	delimiter() Returns the Pattern this Scanner is currently using to match delimiters.
String	findInLine(Pattern pattern) Attempts to find the next occurrence of the specified pattern ignoring delimiters.
String	findInLine(String pattern) Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
String	findWithinHorizon(Pattern pattern, int horizon) Attempts to find the next occurrence of the specified pattern.
String	findWithinHorizon(String pattern, int horizon) Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
boolean	hasNext() Returns true if this scanner has another token in its input.
boolean	hasNext(Pattern pattern) Returns true if the next complete token matches the specified pattern.
boolean	hasNext(String pattern) Returns true if the next token matches the pattern constructed from the specified string.

La classe java.util.Scanner (3)

- Alcuni metodi della classe:

double	**nextDouble**() Scans the next token of the input as a `double`.
float	**nextFloat**() Scans the next token of the input as a `float`.
int	**nextInt**() Scans the next token of the input as an `int`.
int	**nextInt**(int radix) Scans the next token of the input as an `int`.
String	**nextLine**() Advances this scanner past the current line and returns the input that was skipped.
long	**nextLong**() Scans the next token of the input as a `long`.
long	**nextLong**(int radix) Scans the next token of the input as a `long`.
short	**nextShort**() Scans the next token of the input as a `short`.
short	**nextShort**(int radix) Scans the next token of the input as a `short`.
int	**radix**() Returns this scanner's default radix.
void	**remove**() The remove operation is not supported by this implementation of `Iterator`.
Scanner	**reset**() Resets this scanner.



I file

- Un file è una collezione di dati salvata su supporto di memorizzazione di massa
- Un file di dati non è parte del codice sorgente di un programma!
- Lo stesso file può essere letto o modificato da programmi differenti
  - Le funzioni di base sono fornite dal sistema operativo
- Il programma deve conoscere il formato dei dati nel file
- E' IMPORTANTE distinguere tra file di testo e file binari

I file di testo

- Un file di testo contiene linee di testo (ad esempio, in ASCII)
- Ogni linea termina con un carattere di nuova linea (“\n”) o carriage return (“\r”) concatenato con nuova linea
- Esempi:
  - Testo puro (mio_file.txt)
  - Sorgenti di programmi (MiaClasse.java)
  - Documenti HTML (index.html)
  - File di dati testuali gestiti da alcuni programmi (es. std.ics di korganizer)

I file binari

- Un file binario può contenere qualsiasi informazione sotto forma di concatenazione di byte
- Solo il programmatore/progettista sa come interpretarlo
- Diversi programmi potrebbero interpretare lo stesso file in modo diverso (ad esempio, un'immagine)
- Esempi:
  - Programmi compilati (es. MiaClasse.class)
  - File di immagini (es. schermata.gif)
  - File musicali (es. musica.mp3)
- Qualsiasi file può essere trattato come file binario
  - Anche i file di testo!

Sugli stream

- Uno stream è un'astrazione derivata da dispositivi di input o output sequenziale
- Uno stream di input produce un flusso di caratteri
- Uno stream di output riceve un flusso di caratteri "uno alla volta"
- Gli stream non si applicano solo ai file, ma anche a dispositivi di input/output, internet, ecc.
- Un file può essere trattato come uno stream di input o output
- In realtà i file sono bufferizzati per questioni di efficienza

Tre gerarchie di classi in Java

- Per leggere caratteri (tipicamente da file di testo)
  - Reader/Writer
- Per leggere byte (da file binari)
  - InputStream/OutputStream
- L'accesso ai file di testo è stato semplificato in Java 5 mediante l'aggiunta di una classe java.util.Scanner

Leggere un file di testo con FileReader+BufferedReader

- Usando la classe java.io.FileReader combinata con BufferedReader:

```
BufferedReader br = null;

try
{
    br = new BufferedReader(new FileReader("mio_file.txt"));
    while(br.ready())
    {
        System.out.println(br.readLine());
    }
}
catch(IOException e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        if (br != null) br.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}
```

Leggere un file di testo con Scanner

- Usando la classe java.util.Scanner:

```
File f = new File("mio_file.txt");

try
{
    Scanner in = new Scanner(f);

    // finche' esiste una prossima riga
    while(in.hasNext())
        // stampa la riga
        System.out.println(in.nextLine());

    // chiude lo scanner e il file
    in.close();
}
catch(FileNotFoundException e)
{
    e.printStackTrace();
}
```

Costruisce l'oggetto File

Costruisce lo scanner con il file

Itera sulle righe di testo del file

Prossima riga di testo

Chiude scanner e file

La classe File

Field Summary

static String	**pathSeparator** The system-dependent path-separator character, represented as a string for convenience.
static char	**pathSeparatorChar** The system-dependent path-separator character.
static String	**separator** The system-dependent default name-separator character, represented as a string for convenience.
static char	**separatorChar** The system-dependent default name-separator character.

Constructor Summary

File(File parent, String child) Creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname) Creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child) Creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri) Creates a new File instance by converting the given file: URI into an abstract pathname.

Method Summary

boolean	**canExecute**() Tests whether the application can execute the file denoted by this abstract pathname.
boolean	**canRead**() Tests whether the application can read the file denoted by this abstract pathname.
boolean	**canWrite**() Tests whether the application can modify the file denoted by this abstract pathname.
int	**compareTo**(File pathname) Compares two abstract pathnames lexicographically.
boolean	**createNewFile**() Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
static File	**createTempFile**(String prefix, String suffix) Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.

La classe File

boolean	**exists**() Tests whether the file or directory denoted by this abstract pathname exists.
File	**getAbsoluteFile**() Returns the absolute form of this abstract pathname.
String	**getAbsolutePath**() Returns the absolute pathname string of this abstract pathname.
File	**getCanonicalFile**() Returns the canonical form of this abstract pathname.
String	**getCanonicalPath**() Returns the canonical pathname string of this abstract pathname.
long	**getFreeSpace**() Returns the number of unallocated bytes in the partition named by this abstract path name.
String	**getName**() Returns the name of the file or directory denoted by this abstract pathname.
String	**getParent**() Returns the pathname string of this abstract pathname's parent, or `null` if this pathname does not name a parent directory.
File	**getParentFile**() Returns the abstract pathname of this abstract pathname's parent, or `null` if this pathname does not name a parent directory.
String	**getPath**() Converts this abstract pathname into a pathname string.
long	**getTotalSpace**() Returns the size of the partition named by this abstract pathname.
long	**getUsableSpace**() Returns the number of bytes available to this virtual machine on the partition named by this abstract pathname.
int	**hashCode**() Computes a hash code for this abstract pathname.
boolean	**isAbsolute**() Tests whether this abstract pathname is absolute.
boolean	**isDirectory**() Tests whether the file denoted by this abstract pathname is a directory.
boolean	**isFile**() Tests whether the file denoted by this abstract pathname is a normal file.

Scrivere un file di testo con FileWriter+BufferedWriter

- Usando la classe java.io.FileWriter combinata con BufferedWriter:

```
BufferedWriter bw = null;

try
{
    bw = new BufferedWriter(new FileWriter("mio_file.txt"));
    bw.write("Scrivi questo e quest'altro");
}
catch(IOException e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        if (bw != null) bw.close();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}
```

Scrivere un file di testo con PrintWriter

- Si può utilizzare la classe PrintWriter e i metodi corrispondenti:

```
File f = new File("mio_file.txt");

try
{
    PrintWriter out = new PrintWriter(f);

    out.println("Prima riga del file");
    out.print("Seconda riga: ");
    out.print(2);

    // chiude il file
    out.close();
}
catch(FileNotFoundException e)
{
    e.printStackTrace();
}
```

Costruisce l'oggetto File

Costruisce il Writer di file

Scrive due righe di testo

Chiude il file

Esercizio: numerare un file

- Si progetti una classe i cui oggetti sono costruiti a partire da un determinato file di testo f
- La classe espone un metodo numera che:
  1. crea un file il cui nome è dato da quello del file f cui viene concatenata l'estensione ".num"
  2. contiene riga per riga le informazioni contenute nel file f cui viene anteposto il numero di riga
- Ad esempio, date le righe:

```
Questa è la prima riga del file
Questa è la seconda riga del file
```

- salva nel nuovo file:

```
1: Questa è la prima riga del file
2: Questa è la seconda riga del file
```

Scrivere un file di testo formattato

- E' possibile utilizzare la classe java.util.Formatter
- "An interpreter for printf-style format strings"

```
public class FormattaFile
{
    private String nome;
    private int valore;

    public void scrivi(String filename) throws IOException
    {
        Formatter output = new Formatter(filename);
        output.format("%s\t%d", nome, valore);
        output.close();
    }
}
```

Ricorda qualcosa?

Leggere un file di testo formattato

- Essendo il testo nel file formattato usando dei separatori, lo Scanner è in grado di restituire la prossima stringa e il prossimo intero

```
public class FormattaFile
{
    private String nome;
    private int valore;

    public void scrivi(String filename) throws IOException
    {
        Formatter output = new Formatter(filename);
        output.format("%s\t%d", nome, valore);
        output.close();
    }

    public void leggi(String filename) throws IOException
    {
        Scanner input = new Scanner(new File(filename));
        nome = input.next();
        valore = input.nextInt();
    }
}
```

Serializzare un oggetto

Quick Response (QR) code

Stream di bit…

Universal Product Code

Serializzare un oggetto in Java


```
import java.io.*;

public class IlMioOggettoSerializzabile
{
    private String nome;
    private int valore;

    public IlMioOggettoSerializzabile(String nome, int valore)
    {
        this.nome = nome;
        this.valore = valore;
    }

    public void salva(String filename)
    {
        try
        {
            FileOutputStream fos = new FileOutputStream(filename);
            ObjectOutputStream os = new ObjectOutputStream(fos);

            os.writeObject(nome);
            os.writeObject(valore);

            os.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    public static void main(String[] args)
    {
        new IlMioOggettoSerializzabile("dieci", 10).salva("myObject.ser");
    }
}
```

Costruisce uno stream output di file

Costruisce uno stream output di oggetti

Scrive gli oggetti in formato binario

Chiude lo stream

Domanda: che succede se un campo è un riferimento?


```
public class IlMioOggettoSerializzabile
{
    private String nome;
    private int valore;
    private IlMioOggettoSerializzabile next;

    public IlMioOggettoSerializzabile(String nome, int valore)
    {
        this.nome = nome;
        this.valore = valore;
    }

    public void setNext(IlMioOggettoSerializzabile next)
    {
        this.next = next;
    }

    public void salva(String filename)
    {
        try
        {
            FileOutputStream fos = new FileOutputStream(filename);
            ObjectOutputStream os = new ObjectOutputStream(fos);

            os.writeObject(nome);
            os.writeObject(valore);
            os.writeObject(next);

            os.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

    public static void main(String[] args)
    {
        IlMioOggettoSerializzabile o1 = new IlMioOggettoSerializzabile("ciao", 2);
        IlMioOggettoSerializzabile o2 = new IlMioOggettoSerializzabile("dieci", 10);
        o2.setNext(o1);
        o2.salva("myObject.ser");
    }
}
```

Riferimento a un (altro) oggetto

Risposta: dipende se l'oggetto in questione è serializzabile oppure no!

- Nell'esempio, otteniamo:

```
java.io.NotSerializableException: IlMioOggettoSerializzabile
        at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1164)
        at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:330)
        at IlMioOggettoSerializzabile.salva(IlMioOggettoSerializzabile.java:29)
        at IlMioOggettoSerializzabile.main(IlMioOggettoSerializzabile.java:44)
```

- Perché la classe non è "serializzabile"
- Per essere serializzabile, deve implementare l'interfaccia Serializable
  - Un'interfaccia **SENZA METODI**! Perché?
  - E' una sorta di marcatore che indica che la classe ha una determinata caratteristica

Tutto o niente!

- Quando si serializza un oggetto, tutti gli oggetti a cui esso si riferisce (variabili d'istanza) vengono serializzati
- Tutti gli oggetti cui quegli oggetti si riferiscono vengono serializzati
  - Tutti gli oggetti cui quegli oggetti si riferiscono vengono serializzati
    - Tutti gli oggetti cui quegli oggetti si riferiscono vengono serializzati

ecc. ecc.!

Serial Version UID

- E' bene specificare **sempre** un campo static, final long chiamato serialVersionUID
- Usato in fase di (de)serializzazione per verificare se la versione della classe in uso è la stessa usata per serializzare

```
public class IlMioOggettoSerializzabile implements Serializable
{
    private static final long serialVersionUID = -1327935836496038772L;

    private String nome;
    private int valore;
    private IlMioOggettoSerializzabile next;
```

Identificatore univoco di versione

Leggere un oggetto serializzato: Deserializzazione


```
public static IlMioOggettoSerializzabile leggi(String filename)
{
    try
    {
        // apertura del file
        FileInputStream fis = new FileInputStream(filename);
        ObjectInputStream is = new ObjectInputStream(fis);

        // lettura
        Object o1 = is.readObject();
        Object o2 = is.readObject();
        Object o3 = is.readObject();

        // casting
        String nome = (String)o1;
        int valore = (Integer)o2;
        IlMioOggettoSerializzabile next = (IlMioOggettoSerializzabile)o3;

        // creazione dell'oggetto
        IlMioOggettoSerializzabile o = new IlMioOggettoSerializzabile(nome, valore);
        o.setNext(next);

        is.close();

        return o;
    }
    catch (ClassNotFoundException e)
    {
        e.printStackTrace();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }

    return null;
}
```