



UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Laboratorio di Programmazione ad Oggetti

Ph.D. Juri Di Rocco
juri.dirocco@univaq.it
<http://jdirocco.github.io/>





Sommario

- › Leggere valori da riga di comando
- › Scrivere file
- › Leggere file
- › Proprietà



Come leggere input da linea di comando

- › In Java, puoi leggere input dalla linea di comando in una delle possibili alternative:
 - Usare l'oggetto globalmente accessibile `Java Console object ()`.
 - Creare una istanza della classe `Java Scanner class`.
 - Usare `Java's System.in` dato un `InputStream`.



Usando l'oggetto Console

- › È la via più semplice per leggere un input da console. Un esempio di codice:

```
String input = System.console.readLine();  
System.out.println("You typed in: " + input);
```

- › La classe `Console` è molto facile e semplice da usare. Accessibile attraverso l'oggetto universalmente disponibile `System`, quindi non richiede nessun import o la creazione di nuovi oggetti.
- › Il maggior lato negativo è che alcuni Java runtime environments non permettono l'uso di `Console`. Ad esempio Eclipse e the Spring Tool Suite, disabilitano l'uso della `Console`, quindi questa alternativa non funziona in questi IDEs.



User input con la classe the Java Scanner

- › Il modo più comune di prendere gli input da un utente usa la classe `Scanner`. Per usare tale classe:

1. Importare il package `java.util`;
2. Creare un istanza della classe `Scanner`
3. Passere in all costruttore dello scanner.

```
System.out.println("What is your name?");  
Scanner scanner = new Scanner(System.in);  
String name = scanner.nextLine(); System.out.println(name + "  
is a nice name!");
```

- › La classe `Scanner` funziona in tutti gli ambienti.
- › La classe scanner include funzionalità per processare `int`, `float`, `boolean` e `string`-based input.
- › Il maggior lato negativo è che gli sviluppatori sono intimoriti dalla sua semantica non proprio intuitiva.



User input con Java's System.in

- › Nelle prime versioni di JAVA gli sviluppatori potevano ottenere gli input dagli utenti solo concatenando Java I/O classes.

```
InputStreamReader reader = new InputStreamReader(System.in);  
BufferedReader br = new BufferedReader(reader);  
System.out.println("What is your name?");  
String input = br.readLine();  
System.out.println("Your input was: " + input);
```

- › Questa alternativa non è consigliata agli sviluppatori, soprattutto per chi si approccia per la prima volta al linguaggio. Il codice richiede diversi import ed è verboso e non semplice da capire.
- › Evitare di usare questo approccio.



Lavorare con i file

Java fornisce 2 APIs per lavorare con i file.

- › The original `java.io.File` API, disponibile da Java 1.0 (1996).
- › The newer `java.nio.file.Path` API, disponibile da Java 1.7 (2011).

`java.nio.file.Path` fa tutto quello che fa `java.io.File`, ma generalmente in modo migliore :

- › File Features `java.nio.file.Path` supporta i link simbolici, i file attributes ed i metadata support.
- › Migliore utilizzo: ad esempio puoi avere un'eccezione con messaggio di errore più chiaro (no such file, file locked, etc.), invece che `false` un valore di ritorno.



Alcuni esempi di Path

```
public static void main(String[] args) throws URISyntaxException {
    // Java11+ : Path.of()
    Path path = Path.of("/Users/juri/file.txt");
    System.out.println(path);
    path = Path.of("Users","juri","file.txt");
    System.out.println(path);
    path = Path.of("User", "juri").resolve("file.txt");
    // resolve == getChild()
    System.out.println(path);
    path = Path.of(new URI("file:///c:/dev/licenses/windows/readme.txt"));
    System.out.println(path);
    // Java < 11 equivalent: Paths.get()
    path = Paths.get("c:/dev/licenses/windows/readme.txt");
    System.out.println(path); // etc...
}
```




Alcuni esempi di Path

```
public static void main(String[] args) throws URISyntaxException {  
    // Java11+ : Path.of()  
    P  
    S  
    p  
    S  
    p  
    /  
    S  
    p  
    S  
    /  
    p  
    S  
    }  
}
```

Class Paths

java.lang.Object
java.nio.file.Paths

public final class **Paths**
extends **Object**

This class consists exclusively of static methods that return a **Path** by converting a path string or **URI**.

API Note:

It is recommended to obtain a **Path** via the **Path.of** methods instead of via the **get** methods defined in this class as this class may be **deprecate**d in a future release.

Since:

1.7

See Also:

Path



Operazioni comuni sui file

`java.nio.file.Files`

- › Quando si lavora con file e path, si può usare la classe `java.nio.file.Files` che contiene molti metodi statici comuni e per operare con file e cartelle.

- › Verificare se un file esiste

```
Path path = Path.of("Users", "juri", "file.txt");  
boolean exists = Files.exists(path);  
System.out.println("exists = " + exists);
```

- › Prendere la data di ultima modifica

```
Path path = Path.of("Users", "juri", "file.txt");  
FileTime lastModifiedTime = Files.getLastModifiedTime(path);  
System.out.println("lastModifiedTime = " + lastModifiedTime);
```



Operazioni comuni sui file

`java.nio.file.Files`

› Creare file e directory

```
Path newDirectory = Files.createDirectories(path.getParent().resolve("dir"));
System.out.println("newDirectory = " + newDirectory);
Path newFile = Files.createFile(newDirectory.resolve("emptyFile.txt"));
```

› Create temp file e director

```
Path tempFile1 = Files.createTempFile("somePrefixOrNull", ".jpg");
System.out.println("tempFile1 = " + tempFile1);
Path tempFile2 = Files.createTempFile(path.getParent(), "somePrefixOrNull", ".jpg");
System.out.println("tempFile2 = " + tempFile2);
Path tmpDirectory = Files.createTempDirectory("prefix");
System.out.println("tmpDirectory = " + tmpDirectory);
```



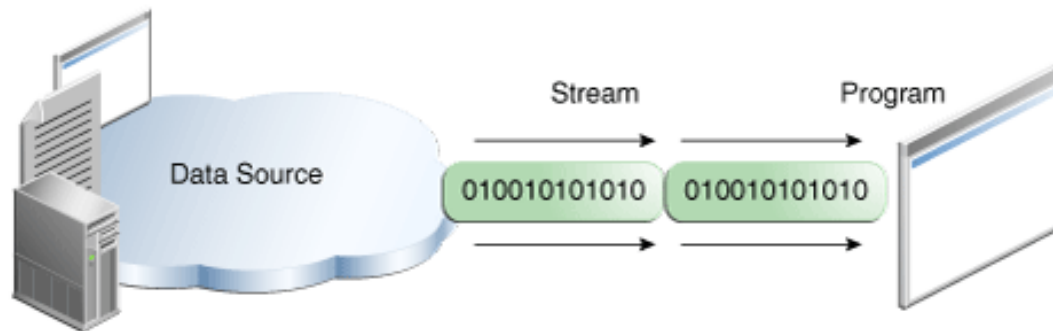
Cosa è

- › In Java, l'Input/Output segue il modello degli stream (o flussi)
- › Stream di I/O rappresenta una sorgente di input oppure una destinazione di output
- › Stream può rappresentare diverse sorgenti e destinazioni inclusi files su disco, dispositivi, altri programmi, array in memoria
- › Stream supportano diversi tipi di dati: bytes, tipi primitivi, caratteri, oggetti
- › Alcuni stream semplicemente passano dati, altri manipolano e trasformano dati
- › Tutti gli stream presentano lo stesso semplice modello
 - Stream è una sequenza di dati
- › Package `java.io` e package `java.nio` versione 2 di Input/Output



Stream input

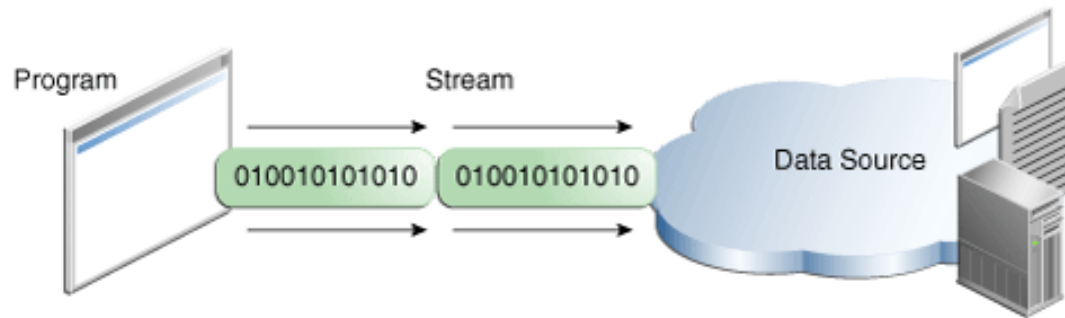
Programma utilizza uno stream di input per leggere dati da una sorgente, un dato alla volta





Stream output

Programma utilizza stream di output per scrivere dati sulla destinazione, un dato alla volta





Stream di byte (1)

- › Utilizzati per eseguire input ed output di bytes (8-bit)
- › Tutte le classi di stream di byte sono figlie di `InputStream` ed `OutputStream`
- › Sono presenti molte classi utilizzabili a seconda delle esigenze



Stream di byte (2)

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class CopyBytes {
    public static void main(String[] args) {
        try (FileInputStream in = new FileInputStream("xanadu.txt");
            FileOutputStream out = new FileOutputStream("outagain.txt");) {
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Legge un byte alla volta e restituisce un intero
(valori 0-255)
-1 se lo stream è terminato

Scrive un byte utilizzando gli 8-bit meno
significativi



Stream di caratteri (1)

- › Java memorizza i valori dei caratteri utilizzando Unicode (16 bit)
- › Stream di caratteri converte automaticamente il formato Unicode al set di caratteri locale (della macchina in cui è in esecuzione)



File Reader – read character

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class CopyCharacters {
    public static void main(String[] args) {
        try (FileReader in = new FileReader("xanadu.txt");
            FileWriter out = new FileWriter("outagain.txt");) {
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Legge un carattere alla volta e restituisce un intero (valori 16-bit 0-65535)
-1 se lo stream è terminato

Scrive un carattere utilizzando i 16-bit meno significativi

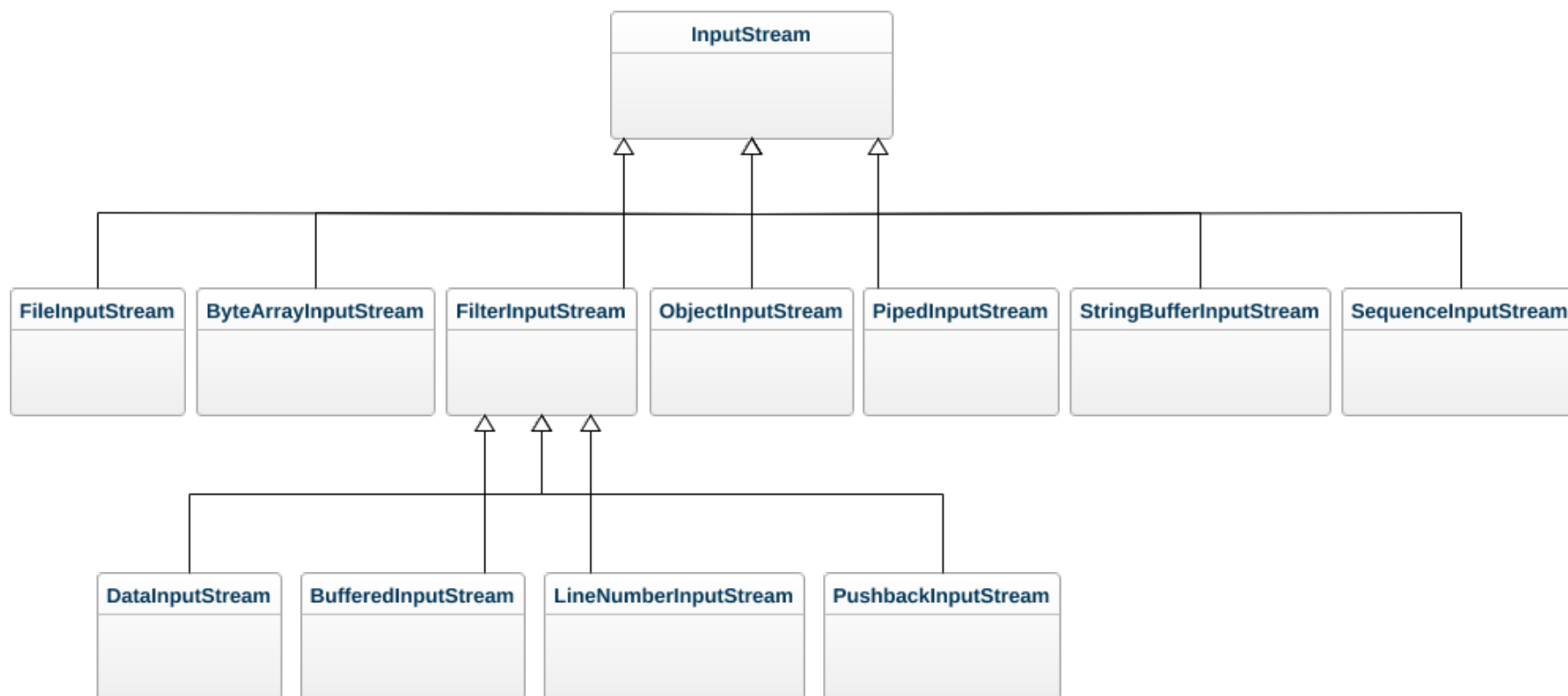


Input/Output orientato alla linea (1)

- › Generalmente l'I/O di caratteri considera unità più grandi di un singolo carattere
- › Unità più comune è la linea
- › Stringa di caratteri che termina con il carattere di fine linea
- › Carattere di fine linea può essere
 - carriage-return/line-feed sequence: `\r\n` (Windows)
 - line-feed: `\n` (GNU/Linux e Mac OS X, tra gli altri)
 - carriage-return: `\r` (Mac OS fino alla versione 9 inclusa)
- › Vengono considerati tutti i caratteri di fine linea per supportare tutti i sistemi operativi

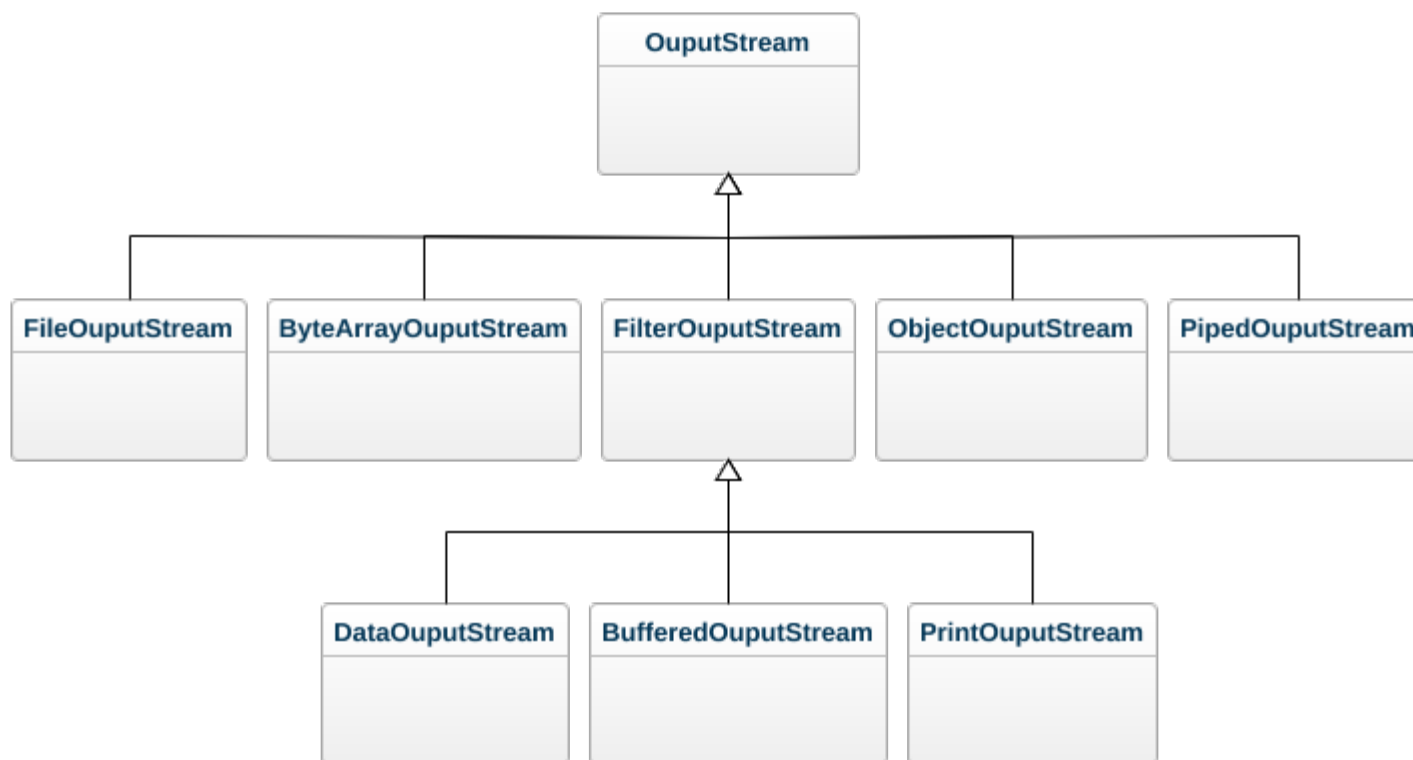


Gerarchia (1)



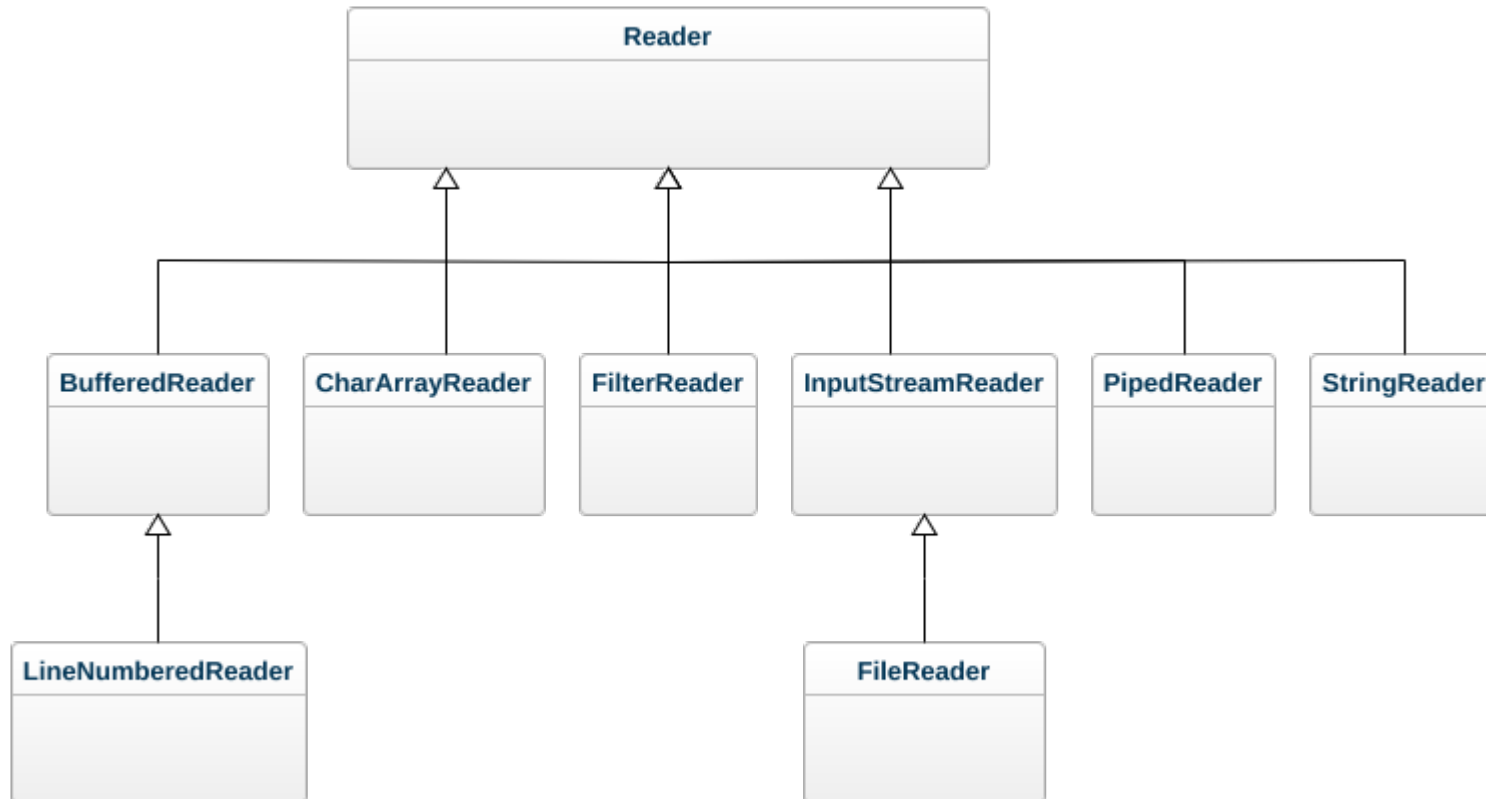


Gerarchia (2)



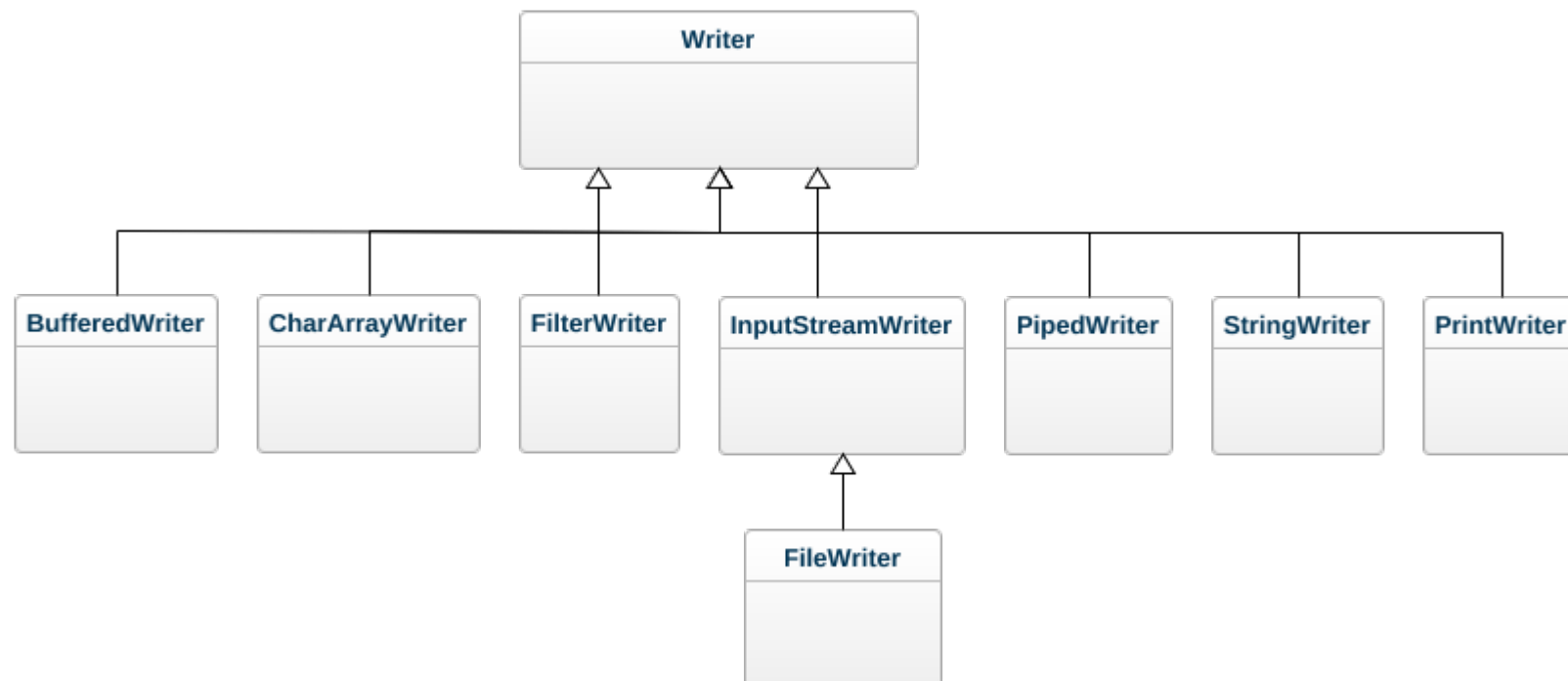


Gerarchia (3)





Gerarchia (4)





Scrivere su file con Files

```
try {
    anotherUtf8File = Files.createTempFile("some", ".txt");
    Files.writeString(anotherUtf8File, "this is my string ää öö üü",
        StandardCharsets.UTF_8,
        StandardOpenOption.CREATE,
        StandardOpenOption.TRUNCATE_EXISTING,
        StandardOpenOption.WRITE);
    System.out.println("anotherUtf8File = " + anotherUtf8File);
    Path oneMoreUtf8File = Files.createTempFile("some", ".txt");

    Files.write(oneMoreUtf8File, "this is my string ää öö
        üü".getBytes(StandardCharsets.UTF_8),
        StandardOpenOption.CREATE,
        StandardOpenOption.TRUNCATE_EXISTING,
        StandardOpenOption.WRITE);
    System.out.println("oneMoreUtf8File = " + oneMoreUtf8File);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Java 11

Java 7



Scivere su file `FileWriter` + `BufferedWriter`

```
try (FileWriter fw = new FileWriter(file);
     BufferedWriter bw = new BufferedWriter(fw)) {
    bw.write(content);
    bw.newLine(); // add new line, System.lineSeparator()
}

try (FileWriter fw = new FileWriter(file, true);
     BufferedWriter bw = new BufferedWriter(fw)) {
    bw.write(content);
    bw.newLine();
}
```

Append mode



Scivere su file `FileWriter` + `BufferedWriter`

```
try (FileWriter fw = Files.newBufferedWriter(path)) {  
    bw.write(content);  
    bw.newLine(); // add new line, System.lineSeparator()  
}  
  
try (FileWriter fw = Files.newBufferedWriter(path,  
        StandardOpenOption.CREATE,  
        StandardOpenOption.APPEND))) {  
    bw.write(content);  
    bw.newLine();  
}
```

Append mode



Scrivere su file `FileOutputStream`

```
try (FileOutputStream fos = new
    FileOutputStream(file)) {
    fos.write(bytes); //ARRAY DI BYTE
}
```



Classe System: Proprietà di sistema (1)

- › Programmi Java vengono eseguiti all'interno di un ambiente che contiene *attributi di sistema*
 - Directory corrente
 - Tipo sistema operativo
 - Versione corrente della JVM
 -
- › Programma Java può anche settare attributi configurabili (*attributi di programma*)
 - Opzioni allo start-up
 -



Classe System: Proprietà di sistema (2)

- › Attributi composto da due parti
 - Nome
 - Valore
- › Esempio
 - Attributo `os.name=Windows XP`
- › Classe `Properties` del package `java.util` gestisce un insieme di coppie chiave/valore
- › Classe `System` gestisce gli attributi di sistema mediante metodi
 - `public static String getProperty(String key)`
 - `public static String getProperty(String key, String default)`
 - `public static Properties getProperties()`



Classe System: Proprietà di sistema (3)

- › Utilizzando opzione `-D` allo startup è possibile poi recuperare la proprietà tramite metodo `getProperty`

```
java -DtestP=Hello Test
```

```
class Test{  
    public static void main(String[] args) {  
        String proprieta = System.getProperty("testP");  
        System.out.println("Proprietà:" + proprieta);  
    }  
}
```

- › E' possibile settare anche le proprietà
 - `public static String setProperty(String key, String value)`
 - `public static setProperty(Properties prop)`



Classe System: Proprietà di sistema (4)

Più comuni

Key	Meaning
"file.separator"	Character that separates components of a file path. This is "/" on UNIX and "\" on Windows.
"java.class.path"	Path used to find directories and JAR archives containing class files. Elements of the class path are separated by a platform-specific character specified in the <code>path.separator</code> property.
"java.home"	Installation directory for Java Runtime Environment (JRE)
"java.vendor"	JRE vendor name
"java.vendor.url"	JRE vendor URL
"java.version"	JRE version number
"line.separator"	Sequence used by operating system to separate lines in text files
"os.arch"	Operating system architecture
"os.name"	Operating system name
"os.version"	Operating system version
"path.separator"	Path separator character used in <code>java.class.path</code>
"user.dir"	User working directory
"user.home"	User home directory
"user.name"	User account name



Classe System: Proprietà di sistema (5)

- › Classe `Properties` fornisce metodi per le seguenti operazioni
 - Caricamento delle coppie chiave/valore da uno stream (es. file) ad un oggetto `Properties`
 - Recupero valore dalla sua chiave
 - Recupero di tutte le chiavi e i loro valore
 - Enumerazione sulle chiavi
 - Salvataggio delle proprietà su uno stream (es. file)



Esempio pratico
