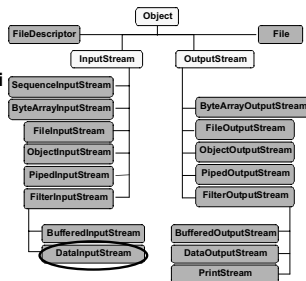


LETTURA DI DATI DA INPUT

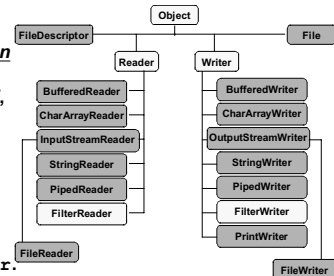
Gli stream di byte consentono già di leggere dati (numeri di vario tipo), tramite la classe `DataInputStream`



LETTURA DI DATI DA INPUT

Sfortunatamente, una tale classe *non esiste per gli stream di caratteri*, tramite i quali è possibile leggere solo stringhe e caratteri.

Può quindi essere utile crearsi una classe `DataReader`.



LETTURA DI DATI DA INPUT

In un precedente esercizio è stata definita la funzione statica `readField`, capace di leggere una sequenza di caratteri da input e trasformarla in stringa:

```
static public String readField(Reader in){
    ...
}
```

Ora si tratta di costruire una nuova classe `Reader`, in cui tale funzione assuma la forma di metodo, e costituisca la base per leggere ogni tipo di dato (int, float, double, stringhe...)

LA FUNZIONE `readField`

```
static public String readField(Reader in){
    StringBuffer buf = new StringBuffer();
    boolean nospace = true;
    try { while (in.ready() && nospace){
        char ch = (char)in.read();
        nospace = (ch!=' ');
        if (nospace) buf.append(ch);
    }
    } catch (IOException e){
        System.out.println("Errore di input");
        System.exit(4);
    }
    return buf.toString();
}
```

DataReader: IL PROGETTO

Dove collocarla nella gerarchia?

- se si fa derivare `DataReader` da `FileReader` si può usarla solo con i file → troppo limitativo
- far derivare `DataReader` da `InputStreamReader` non è praticamente possibile, perché il costruttore di `InputStreamReader` richiede un `InputStream` (non un `Reader`), quindi scegliendo questa via la classe non sarebbe usabile con i `Reader`, e in particolare con i `FileReader` → inutile

Una scelta possibile è creare una classe “wrapper” che accetti come argomento un `InputStreamReader`.

DataReader: IL PROGETTO

Quindi, `DataReader`:

- discenderà direttamente da `Reader`
- costruirà i suoi oggetti a partire da un oggetto `InputStreamReader`

```
class DataReader extends Reader {
    ...
    public DataReader(InputStreamReader in) {
        ...
    }
    ...
}
```

DataReader: IL PROGETTO

Problema: Reader è una classe astratta

- derivare DataReader direttamente da Reader implica *implementare i due metodi astratti*

```
public void close() throws IOException;
public int read(char buf[],
    int off, int len) throws IOException;
```

Come farlo?

- delegando ogni azione all'oggetto InputStreamReader che è dentro al DataReader

DataReader: IL PROGETTO

Architettura di un oggetto DataReader

```
public class DataReader extends Reader {
    InputStreamReader in;
    public DataReader(InputStreamReader in) {
        this.in = in; }
    ...
}
```

Ogni operazione richiesta al DataReader deve essere delegata all'oggetto interno.



DataReader: IL PROGETTO

Implementazione dei metodi astratti tramite delegazione:

```
public void close() throws IOException {
    in.close();
}

public int read(char b[], int off, int len)
    throws IOException {
    return in.read(b, off, len);
}
```

DataReader: IL PROGETTO

Implementazione del metodo readString (versione metodo della funzione readField):

```
public String readString()
    throws IOException {
    StringBuffer buf = new StringBuffer();
    boolean go = true;
    do {
        int x = in.read(); // DELEGAZIONE
        if (x!=-1) { // end-of-file not reached
            char ch = (char)x;
            go = (ch!=' ' && ch!='\n' && ch!='\t' && ch!='\r');
            if (go) buf.append(ch);
        } else go=false;
    } while(go);
    return buf.toString();
}
```

Ampliato il set di caratteri di separazione

DataReader: IL PROGETTO

Gli altri metodi:

```
public int readInt() throws IOException {
    return Integer.parseInt(readString());
}

public float readFloat() throws IOException {
    return Float.parseFloat(readString());
}

public double readDouble() throws IOException {
    return Double.parseDouble(readString());
}

public boolean readBoolean() throws IOException {
    return readString().equals("true");
}

public char readChar() throws IOException {
    return readString().charAt(0);
}
```

ESEMPIO 1 (lettura da file di testo)

```
public class EsDataReader1 {
    public static void main(String args[]) {
        FileReader fin = null;
        try { fin = new FileReader("Prova.txt"); }
        catch (FileNotFoundException e) {
            System.out.println("File non trovato");
            System.exit(1);
        }
        DataReader dr = new DataReader(fin);
        ...
        // ipotesi: il file di testo contiene nell'ordine
        // un float, un boolean, un double, un char e un
        // int, separati da uno e un solo spazio
    }
}
```

ESEMPIO 1 (lettura da file di testo)

```
...
try {
    float f2 = dr.readFloat();
    boolean b2 = dr.readBoolean();
    double d2 = dr.readDouble();
    char c2 = dr.readChar();
    int i2 = dr.readInt();
    dr.close();
    System.out.println(f2 + ", " + b2 + ", "
        + d2 + ", " + c2 + ", " + i2);
}
catch (IOException e){
    System.out.println("Errore di formato in input");
    System.exit(2);
}
}
```

ESEMPIO 2 (lettura da console)

```
public class EsDataReader2 {
    public static void main(String args[]){
        DataReader consoleInput = new DataReader(
            new InputStreamReader(System.in) );
        System.out.println("Battere un float, un boolean,"
            + " un double, un char e un int ");
        try { // separati da UNO E UN SOLO spazio
            float f2 = consoleInput.readFloat();
            boolean b2 = consoleInput.readBoolean();
            double d2 = consoleInput.readDouble();
            char c2 = consoleInput.readChar();
            int i2 = consoleInput.readInt();
        }
        catch (IOException e){...}
    }
}
```