Questa presentazione non ha un titolo

In Java **NON esistono** tipi interi unsigned (senza segno)!

Stringhe e bytes

Un byte in Java è un tipo primitivo a 8 bit con segno (da -128 a 127). Quando lavori con più byte, devi capire come interpretarli.

Il tipo byte in Java è:

- Signed (con segno)
- 8 bit
- Range: **-128 a 127**

```
package edu.avo;

public class spiegazioneStringheByte {
    public static void main(String[] args) {
      // da String a byte[]
      String messaggio = "Ciao";
      byte[] bytes = messaggio.getBytes();

      // da byte[] a String
      String ricostruita = new String(bytes);
      System.out.println("Stringa ricostruita: " +
      ricostruita);


}
```

Tipo byte in Java

Un byte in Java è un tipo primitivo a 8 bit con segno (da -128 a 127). Quando lavori con più byte, devi capire come interpretarli.

Il tipo byte in Java è:

- Signed (con segno)
- 8 bit
- Range: **-128 a 127**

```
package edu.avo;

public class spiegazioneByte {
    public static void main(String[] args) {
        byte b = 10;
        int valore_b = b;
        System.out.println(valore_b);

        b = -10;
        valore_b = b;
        System.out.println(valore_b);

        //vediamo problema del segno e della propagazione
        //serve cast esplicito perché andiamo fuori dal
range
        b = (byte) 124;
        valore_b = b;
        System.out.println(valore_b);

        //& 0xFF serve per convertire
        // il byte (con segno) in un valore senza segno.
        valore_b = b & 0xFF;
        System.out.println(valore_b);

    }
}
```

Combinare i byte

Se ricevi dati binari come:

Un numero intero a 16 bit (2 byte)

Un numero intero a 32 bit (4 byte)

Un float (4 byte)

Si estraggono e combinano i byte

Se hai un numero su 4 byte (32 bit), devi shiftare ogni byte in base alla sua posizione:

```
package edu.avo;

public class spiegazioneDueByte {
    public static void main(String[] args) {
        // Creiamo due byte
         byte byte1 = 1;  // 0X12
         byte byte2 = 24;  // 0x34

         System.out.println("Byte 1: " + byte1 + " decimale)");
         System.out.println("Byte 2: " + byte2 + " decimale)");
         System.out.println();

         //e se voglio rappresentare numeri non da 0 a 255 ma

         // Metodo 1: Big-endian (byte più significativo prima)
         // byte 1 00000001 diventa 00000001 00000000
         // byte 2 00011000 diventa 00000000 00011000
         int byteMsb = (byte1 & 0xFF) << 8;
         int byteLsb = byte2 & 0xFF;
         System.out.println("msb: " + byteMsb);
         System.out.println("lsb: " + byteLsb);
         // operatore | combina i due byte byte
         // 00000001 0000000
         int bigEndian = ((byte1 & 0xFF) << 8) | (byte2 & 0xFF);
         System.out.println("Big-endian (0x1234): " + bigEndian);

    }
}
```

perché & 0XFF
Maschera di bit

Cos'è l'estensione del segno?

Quando converti un tipo più piccolo (es: byte) in uno più grande (es: int), Java copia il bit di segno in tutti i nuovi bit aggiunti.

byte bytePositivo = 50; // In binario: 00110010 int intPositivo = bytePositivo;

Binario int: 00000000000000000000000000110010

// Esempio 2: Numero NEGATIVO (qui succede l'estensione del segno!) byte byteNegativo = -50; // In binario: 11001110 int intNegativo = byteNegativo;

Binario int: 11111111111111111111111111001110

Si usa sempre & 0xFF quando

- converti byte in int e vuoi valori da 0 a 255
- Combini byte per formare numeri più grandi

Tipo byte in Java

Altri linguaggi (come C# con `byte` unsigned) non hanno questo problema.

In Python è MOLTO più semplice perché fornisce metodo già presente librerie base (si dice built-in)

Nessuno shift manuale, nessun problema di segno!

In C dipende molto da cosa fai, ma hai più controllo e più problemi potenziali. Dipende anche da architettura (Intel big endian, altri processori little endian)

```
# no problema segno
# In Python, bytes sono SEMPRE da 0 a 255
b = bytes([200])
print(b[0])  # 200 (non -56 come in Java!)

# Hai 4 byte
bytes_data = bytes([0x12, 0x34, 0x56, 0x78])

# Conversione diretta!
numero = int.from_bytes(bytes_data, byteorder='big')
print(numero)  # 305419896

# O little-endian
numero_little = int.from_bytes(bytes_data,
byteorder='little')
print(numero_little)  # 2018915346

umero = 305419896 # Converti in 4 byte bytes_data =
numero.to_bytes(4, byteorder='big') print(bytes_data) #
b'\x12\x34\x56\x78'
```

TFTP

Tipi di Pacchetti:
- RRQ (1): Read Request - richiesta di lettura
- WRQ (2): Write Request - richiesta di scrittura
- DATA (3): blocco dati (max 512 byte)
- ACK (4): acknowledgment
- ERROR (5): segnalazione errore

Meccanismo Lock-Step:
- Ogni DATA packet viene ACKato prima di inviare il successivo
- Block numbers partono da 1 e aumentano sequenzialmente
- DATA < 512 byte = fine trasferimento
- ACK(0) = risposta iniziale a WRQ

TFTP - sendData op Code 3

[0][3][blockNumber MSB][blockNumber LSB][dati…]

block number occupa due byte

public void sendData(int blockNumber, byte[] data, InetAddress address, int port) throws IOException {

… questo metodo deve ricostruire array di bytes secondo lo schema [0][3][blockNumber MSB] [blockNumber LSB][dati…]

mi arriva int blockNumber che devo portare su …
mi arriva array di byte data che rappresenta il blocco del file
usare System.arraycopy(src, srcPos, dest, destPos, length);
crea DatagramPacket
spedirlo

TFTP - sendAck op Code 4

[0][4][blockNumber MSB][blockNumber LSB]

block number occupa due byte

public void sendAck(int blockNumber, InetAddress address, int port)

… questo metodo deve ricostruire array di bytes secondo lo schema [0][4][blockNumber MSB] [blockNumber LSB]

TFTP - sendError op Code 5

[0][5][0][errorCode 1 byte][messaggio error]

block number occupa due byte

public void sendError(int errorCode, InetAddress address, int port)

… questo metodo deve ricostruire array di bytes secondo lo schema [0][5][0][errorCode 1 byte][messaggio error]
ho errorCode… devo recuperare il messaggio che dato codice sarà sempre lo stesso secondo RFC..
che facciamo? Ho un intero devo recuperare la stringa errore messaggio. Recuperata la stringa, la converto in bytes e uso System.arraycopy(src, srcPos, dest, destPos, length);