

For English language systems java's primitive char type's value is an index (long) into the unicode table of character symbols.

Using an index allows the table to be changed for different languages.

Character class is a “wrapper” class for primitive char (see API docs)

Example Character methods:

```
public static boolean isDigit(char ch)
```

```
    // is the specified character is a digit.
```

```
public static boolean isLetter(int codePoint)
```

```
    // is specified character (Unicode code point) a letter.
```

```
public static boolean isLowerCase(char ch)
```

```
    // is the specified character is a lowercase character.
```

```
public static Character valueOf(char c)
```

```
    // returns a Character instance representing the char value.
```

Consider the following demonstration and its output

```
public class CharAndInt {
    public CharAndInt() {
        char c;
        int i;

        System.out.println("index \t as char \t char as int");
        for(int j = 0; j < 128; j++) {
            c = (char) j;          // cast is needed
            i = c;                // cast is not needed
            System.out.println(j + "\t" + c + "\t\t" + i);
        }
    }
}
```

**output ...**

index	as char	char as int
0		
...		
8		8
9		9
10		
	10	
11		
	11	
...		
31		31
32		32
33	!	33
34	"	34
...		

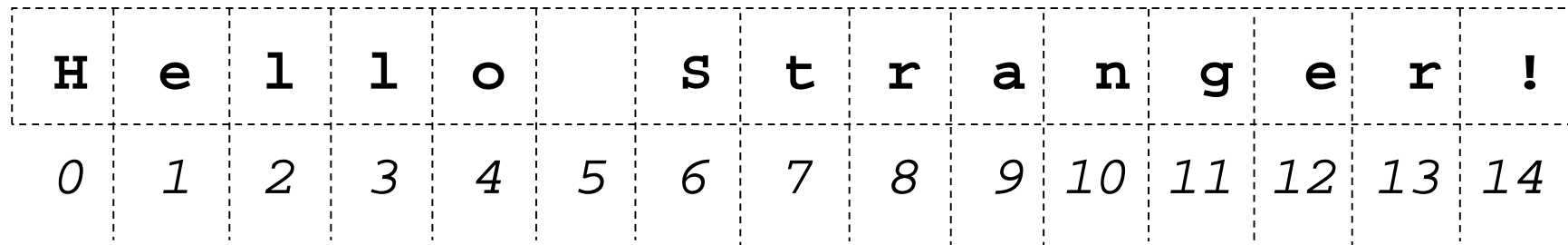
index	as char	char as int
...		
48	0	48
49	1	49
...		
64	@	64
65	A	65
66	B	66
...		
96	`	96
97	a	97
98	b	98
...		
121	y	121
122	z	122
123	{	123
...		

# Strings

String class represents a sequence of character values and has methods that can be applied to string objects.

A string is like an array of characters.

```
String sayHi = "Hello Stranger!";
```



```
sayHi.length();           returns the number of characters in string  
                           returns 15  
sayHi.toLowerCase();     returns a string -- all lower case  
sayHi.toUpperCase();     returns a string -- all upper case  
String SAYHI = sayHi.toUpperCase();  
sayHi.charAt(index);     returns the char value a position index  
sayHi.charAt(4);         returns 'o'
```

`String str2, str1 = "Hello Stranger!";` *Strings && input*

5

`int index`

`str1.equals(str2);` true when str1 is equal to str2

`str1.equalsIgnoreCase(str2);` true when str1 equals str2  
regardless of case

`str1.compareTo(str2);` does lexicographic comparison  
returns 0 when str1 equals str2, is negative when str1 < str2,  
and is positive when str1 > str2 -- use to sort arrays of Strings

`str1.substring(index);` returns substring in str1 from  
position index to end

`str2 = str1.substring(3);` returns "lo Stranger!"

`str2 = str1.substring(8,13);` returns "range"  
start at 8, extract string up, but ! 13

`index = str1.indexOf("range");` returns 8, -1 if substring ! found

```
// insertion sort with strings
const int MAX = 5;
String[] str = new String[MAX];
String temp;

// assume 5 strings in str[0] .. str[4]

for(i = 0; i < MAX - 1; i++)
    for(j = i+1; j < MAX; j++)
        if (str[i].compareTo(str[j]) > 0) {
            temp = str[i];
            str[i] = str[j];
            str[j] = temp; }
}
```

6 exchanges  
needed to sort

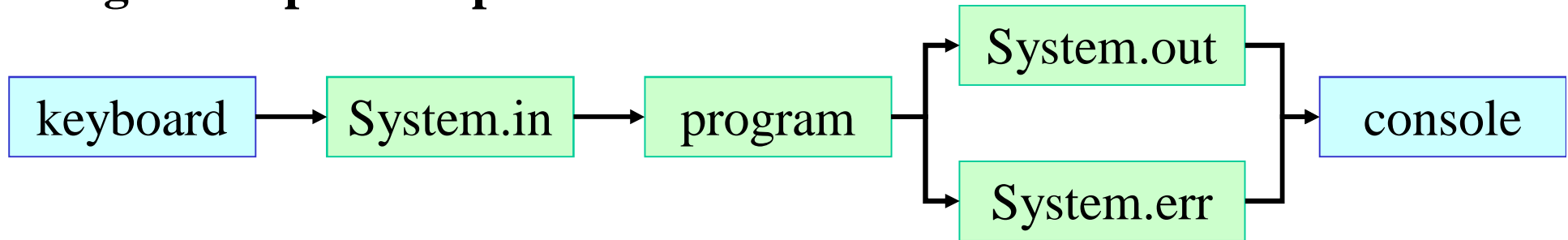
4 needed with  
selection sort

str[0]	hello					bye			
str[1]	there								
str[2]	stranger								
str[3]	bye						hello		
str[4]	now								
temp					hello				
i	0								
j	1	2	3					4	
compareTo	F		F	T					F

# Program Input / output

*Strings && input*

7



Java Virtual Machine defines 3 I/O streams: in, out, and err

System.in      keyboard    InputStream

System.out     consolePrintStream

System.err     consolePrintStream

streams process (channel) data byte at a time

Standard output is easy

`System.out.print(aString)` Or `System.out.println(aString)`

Error output can also use `print()` and `println()` methods

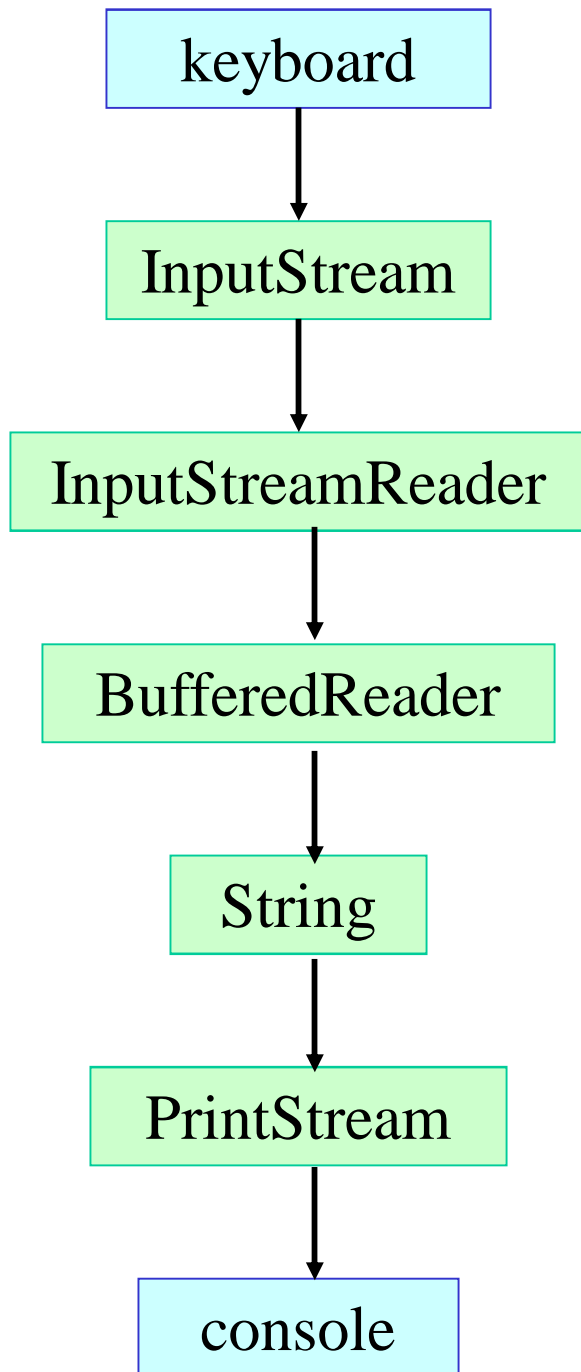
Input is difficult -- need to convert bytes to typed values.

Java and other languages have formatting strings that convert input and assign to variables in a single call. Print tables, formatted reports.

`% [argument_index$] [flags] [width] [.precision] conversion`

field	purpose
argumentIndex	argument index -- rarely used
flags	alignment - left + include sign 0 zero padded
width	number of spaces
precision	number decimal digits + "dot"
conversion	type to convert to string: d decimal f float g general o octal h hex c char

```
System.out.println("Pi = %5.3f", Math.PI);
```



```
# import java.io.*; Strings && input

public class Hello {

public static void main (String args)
  throws IOException {

  BufferedReader input = new
  InputStreamReader(System.in);

  System.out.print("Enter name: ");

  String name = input.readLine();

  System.out.println("Hi " + name);
  }
}
```

IO Exception occurs when system detects problem doing IO. (Exceptions in later lectures)

BufferedReader constructor works because of class inheritance (later topic).

## Reading input one variable's value per line

get the line into string with `BufferedReader`'s `readLine()` method  
convert string value to variable's type and assign to variable

```
import java.io.*;

public class IOexample {

    public static void main (String args[]) throws IOException {
        BufferedReader input =
            new BufferedReader(new InputStreamReader(System.in));

        char ch;
        int i;
        double x;
        String str;

        System.out.print("Enter a char:  ");
        str = input.readLine();
        ch = str.charAt(0);
        System.out.println("char is " + ch);

        System.out.print("Enter a string:  ");
        str = input.readLine();
        System.out.println("string is : " + str);
    }
}
```

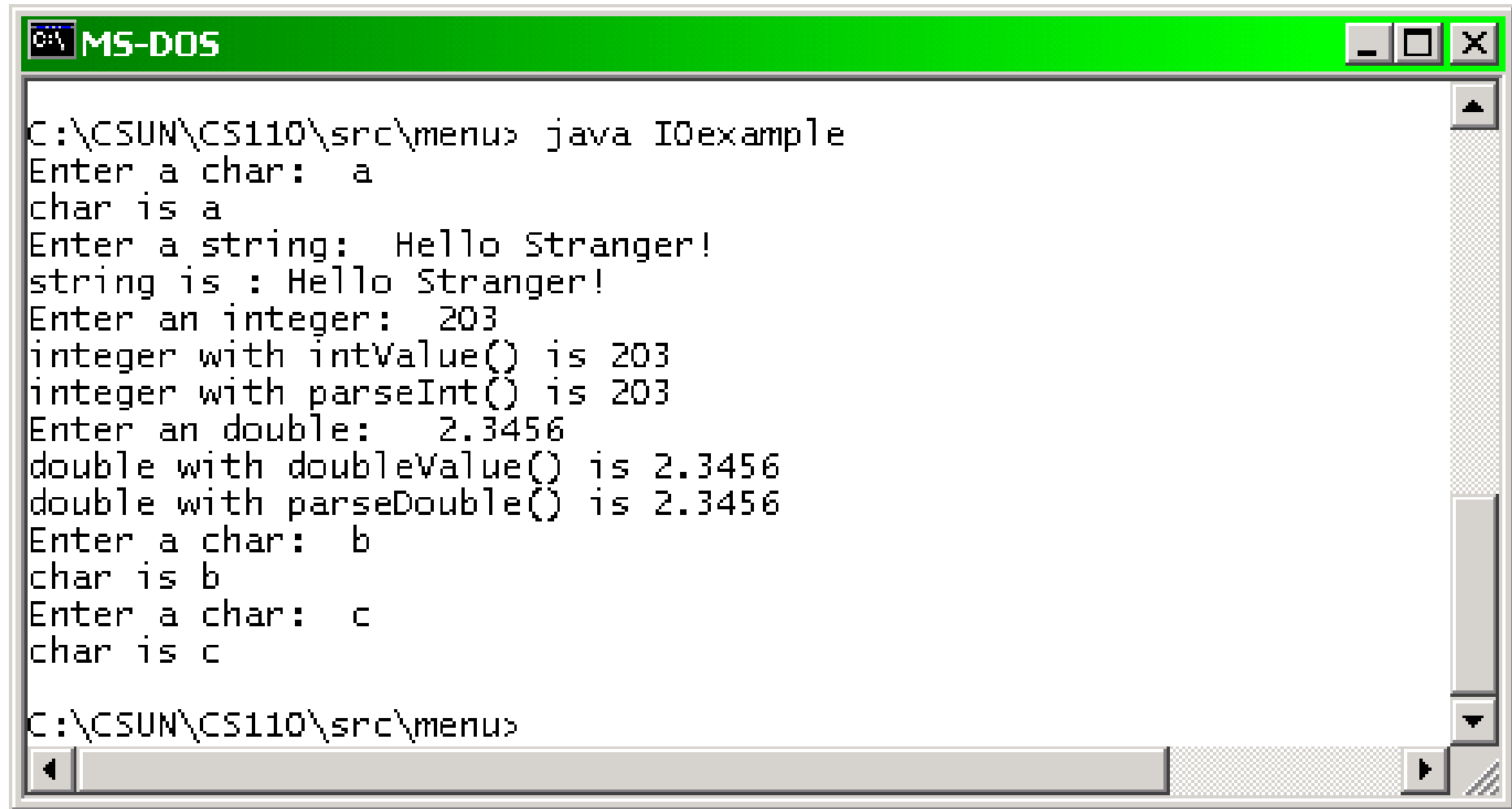
```
System.out.print("Enter an integer:  ");
str = input.readLine();
i = new Integer(str).intValue();
System.out.println("integer with intValue() is " + i);
i = Integer.parseInt(str);
System.out.println("integer with parseInt() is " + i);

System.out.print("Enter an double:  ");
str = input.readLine();
x = new Double(str).doubleValue();
System.out.println("double with doubleValue() is " + x);
x = Double.parseDouble(str);
System.out.println("double with parseDouble() is " + x);

System.out.print("Enter a char:  ");
ch = (char) (input.read());
System.out.println("char is " + ch);

System.out.print("Enter a char:  ");
// need to clear out end of input stream from last read.
str = input.readLine(); // better than skip
ch = (char) (input.read());
System.out.println("char is " + ch);
```

```
}
}
```



```
C:\CSUN\CS110\src\menu> java IOexample
Enter a char:  a
char is a
Enter a string: Hello Stranger!
string is : Hello Stranger!
Enter an integer: 203
integer with intValue() is 203
integer with parseInt() is 203
Enter an double: 2.3456
double with doubleValue() is 2.3456
double with parseDouble() is 2.3456
Enter a char:  b
char is b
Enter a char:  c
char is c

C:\CSUN\CS110\src\menu>
```

Using BufferedReader's readLine() with a menu interface

```
// example menu in java

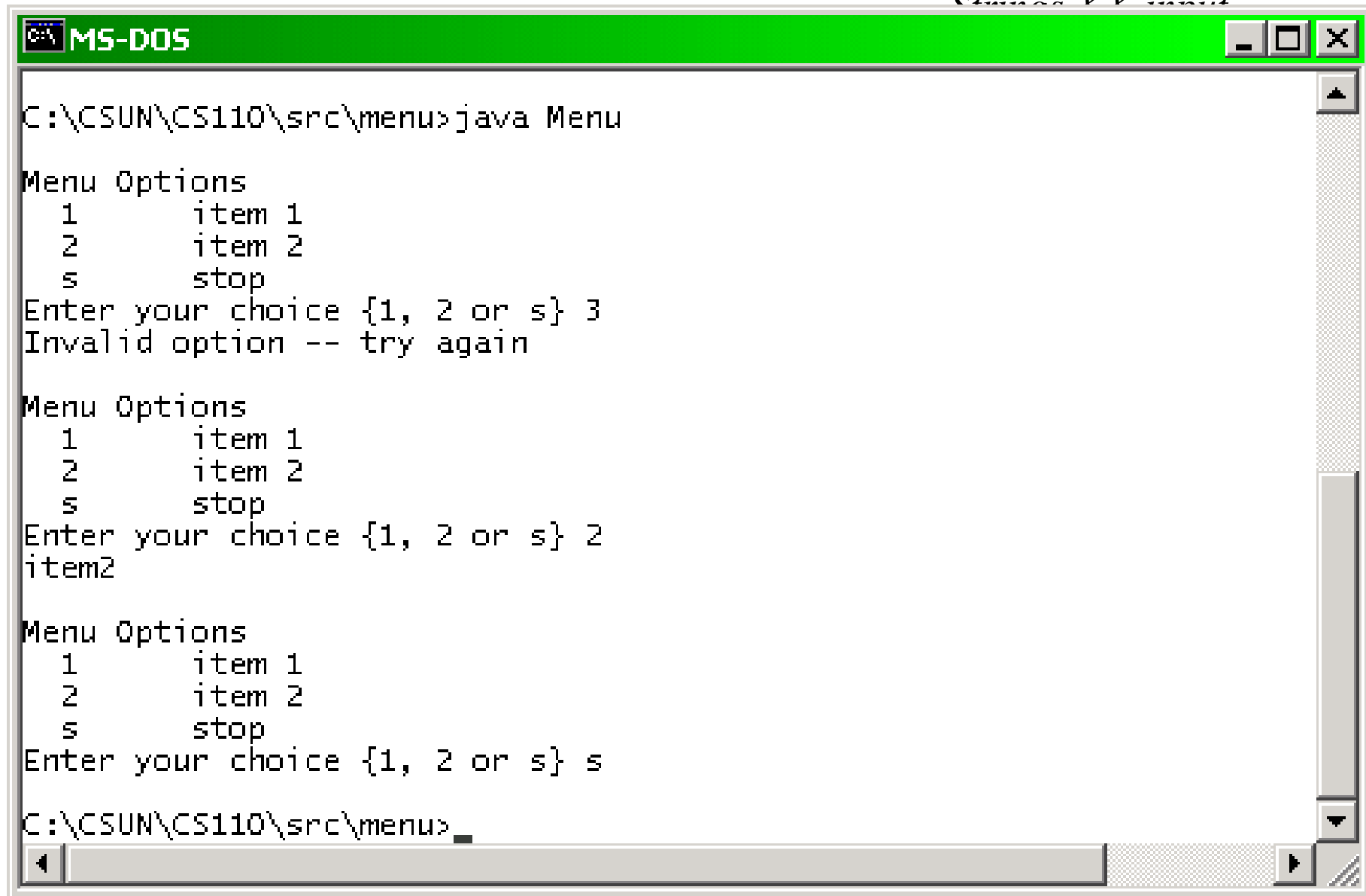
import java.io.*;

public class Menu {

    public static void main (String args[]) throws IOException {
        Menu app = new Menu();
        app.doMenu();
    }

    public void item1() {System.out.println("item1");}
    public void item2() {System.out.println("item2");}
```

```
public void doMenu() throws IOException {
    BufferedReader input =
        new BufferedReader(new InputStreamReader(System.in));
    boolean done = false;
    char answer;
    do {
        System.out.println();
        System.out.println("Menu Options ");
        System.out.println("  1 \t item 1");
        System.out.println("  2 \t item 2");
        System.out.println("  s \t stop");
        System.out.print("Enter your choice {1, 2 or s} ");
        answer = (input.readLine()).charAt(0);
        switch (answer) {
            case '1' : item1(); break;
            case '2' : item2(); break;
            case 's' : done = true; break;
            default  : System.out.println(
                "Invalid option -- try again");
        }
    }
    while (!done);
}
```



```
MS-DOS
C:\CSUN\CS110\src\menu>java Menu

Menu Options
 1      item 1
 2      item 2
 s      stop
Enter your choice {1, 2 or s} 3
Invalid option -- try again

Menu Options
 1      item 1
 2      item 2
 s      stop
Enter your choice {1, 2 or s} 2
item2

Menu Options
 1      item 1
 2      item 2
 s      stop
Enter your choice {1, 2 or s} s

C:\CSUN\CS110\src\menu>
```

Reading many values per line

get the line into string with `BufferedReader`'s `readLine()` method

construct a `StringTokenizer` with input string

get and convert each token to variable's type and assign to variable

A string can consists of tokens separated by delimiters.

"The quick red fox jumped over the lazy brown dog."

10 tokens, words, separated by blanks " " as delimiters.

constructors

```
public StringTokenizer(String str) // blank is delimiter
public StringTokenizer(String str, String delimiters)
    new StringTokenizer(aString, " ."); //blank && .
```

methods

```
public boolean hasMoreTokens() // true when ! empty
public int countTokens() // number of tokens
public String nextToken() // next token available
```

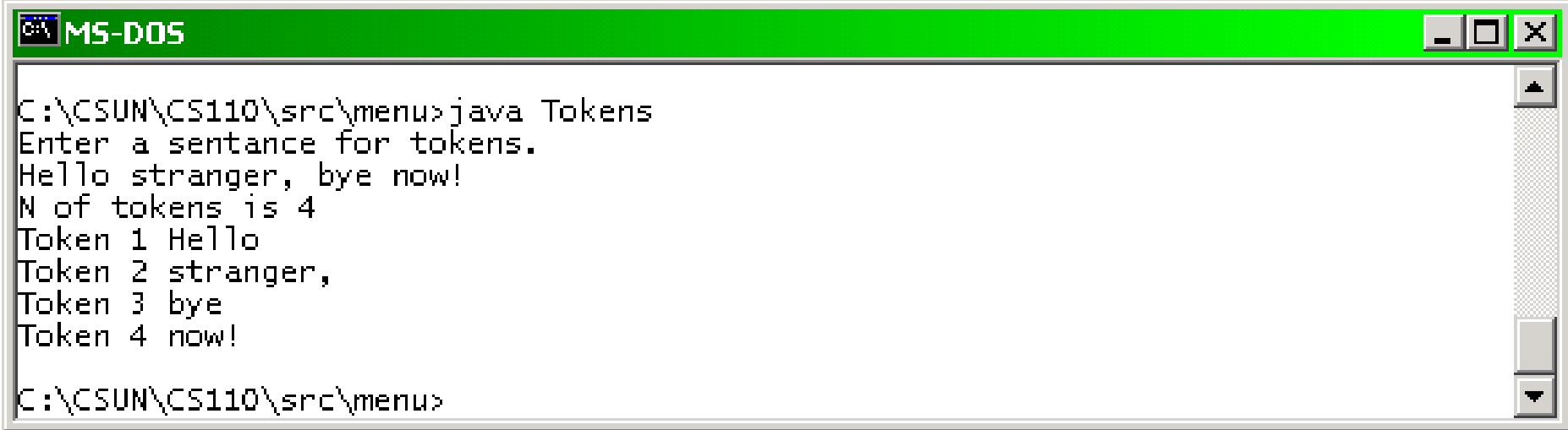
```
import java.io.*;
import java.util.*; // needed for StringTokenizer

public class Tokens {

    public static void main (String args[]) throws IOException {
        BufferedReader input =
            new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer tokens;
        int i = 0;

        System.out.println("Enter a sentence for tokens.");
        tokens = new StringTokenizer(input.readLine());
        System.out.println("N of tokens is " + tokens.countTokens());
        while (tokens.hasMoreTokens())
            System.out.println("Token " + ++i + " " +
                tokens.nextToken());
    }
}
```

Could you use countTokens() value and a for loop instead ?



```
MS-DOS
C:\CSUN\CS110\src\menu>java Tokens
Enter a sentence for tokens.
Hello stranger, bye now!
N of tokens is 4
Token 1 Hello
Token 2 stranger,
Token 3 bye
Token 4 now!

C:\CSUN\CS110\src\menu>
```

Setting the delimiters string in the constructor

```
tokens = new StringTokenizer(input.readLine(), " ,.!");
```

would generate tokens without the ",", "." or "!" symbols

Each extracted token must be converted to correct type before assigned.

Scanner class helps console and file input.

replaces BufferedReader, has some StringTokenizer features.

The default whitespace delimiters used by a scanner are blank spaces, tabs, line breaks – prints as whitespace  
`aScanner.useDefault("...")` can set delimiter values.

Constructors:

```
Scanner(InputStream source);
```

```
Scanner aScanner = new Scanner(System.in);
```

```
Scanner(File source);
```

```
Scanner aScanner = new Scanner(new File("myNumbers"));
```

`aScanner.close()` will close the current Scanner's stream.

`aScanner.next<Type>()` will return the next value of `next<Type>` if the token can be converted to `<Type>`

```
anInt = aScanner.nextInt();           // reads next int
aFloat = aScanner.nextFloat();        // reads next float
aBoolean = aScanner.nextBoolean();    // reads next boolean
aString = aScanner.next();            // reads next string
aString = aScanner.nextLine();        // reads rest of line
```

`aScanner.hasNext<Type>()` is true if the next token can be converted to `<Type>`

```
if (aScanner.hasNextInt())
    anInt = aScanner.nextInt();

if (aScanner.hasNextFloat())
    aFloat = aScanner.nextFloat();
```

`aScanner.hasNext()` is true if there is a next token – not end of line or input stream.