Basic Input and Output

Methods for reading input and writing output.

James Brucker

Display output

```
System.out.println("I'm a string" + " again");

System.out.print("apple");
System.out.print("banana\n");
System.out.print("grape");
```

```
I'm a string again
applebanana
grape
```

Input

System.in can only read bytes. Not very useful.

```
int c = System.in.read( ); // read 1 byte
byte[] b;
System.in.read( b );  // array of bytes
```

Use a Scanner to read input as int, double, String, etc.

```
Scanner console = new Scanner(System.in);
String word = console.next();
String line = console.nextLine();
int number = console.nextInt();
double x = console.nextDouble();
```

Console Output: `print`

`System.out` is a PrintStream object.

It has a "**`print`**" method to output (display) :

- any primitive data type
- a String
- any Object

```
int a = 100;
System.out.print("a = "); // print a string
System.out.print(a);      // print an int
System.out.print('\t');   // print a TAB char
System.out.print("Square Root of 2 = ");
System.out.print(Math.sqrt(2)); // print double
```

```
a = 100   Square Root of 2 = 1.4142135623730951
```

Console Output: `println`

`println` is like `print`, but after printing a value it also outputs a newline character to move to start of next line.

`println` can output:

- any primitive type
- a String
- any Object: it automatically calls object.toString()

```
System.out.print("a = "); // print a string
System.out.println(a);    // print an int
System.out.println();  // empty line
System.out.println(1.0/3.0); // print double
```

```
a = 100

0.333333333333333
```

More on `print` and `println`

To print several values at once, if the *first value is a String*, you can "join" the other values using +

```
System.out.println("a = " + a);
```

Is the same as:

```
System.out.print("a = ");      // print a string
System.out.println(a);        // print an int
```

```
a = 100
```

Printing an Object

If the argument is an object, Java will call the object's toString() method and print the result.

```
Date now = new Date( );
System.out.println( now );
   // invokes now.toString()
```

Common Error

ERROR:

```
double angle = Math.toRadians(45);
double x = Math.sin(angle);
System.out.println("sin(" , angle ,") = " , x);
```

mus use + not comma

Formatted Output: `printf`

Creating nice output using println can be difficult.

```
public class SalesTax {
   public static final double VAT = 0.07;  // 7% tax
   public static void showTotal( double amount) {
      double total = amount * ( 1.0 + VAT );
      System.out.println("The total including VAT is "
      +total+" Baht");
   }
  public static void main( String [] args ) {
     showTotal(10000);
     showTotal(95);
   }
}
```

```
The total including VAT is 10700.0 Baht

The total including VAT is 104.86 Baht
```

printf

Java 1.5 added a "printf" statement similar to C:

```
 public static void showTotal( double amount) {
     double total = amount * ( 1.0 + VAT );
     System.out.printf(
     "The total including VAT is %8.2f Baht", total);
}
public static void main( String [] args ) {
  showTotal(10000);
  showTotal(95);
}
```

Format: output a float (%f) using 8 characters with 2 decimal digits

```
The total including VAT is 10700.00 Baht

The total including VAT is   104.86 Baht
```

printf Syntax

The syntax of printf is:

```
System.out.printf(Format_String, arg1, arg2, ...);
```

or (no arguments):

```
System.our.printf(Format_String);
```

The *Format_String* can contain text and format codes. Values of arg1, arg2, are substituted for the format codes.

```
int x = 100, y = 225;
System.out.printf("The sum of %d and %d is %6d\n",
          x, y, x+y );
```

%d is the format code to output an "int" or "long" value.

%6d means output an integer using exactly 6 digit/spaces

printf Syntax

Example: print the average

```
int x = 100, y = 90;
System.out.printf("The average of %d and %d is %6.2f\n",
            x, y, (x+y)/2.0 );
```

%6.2f means output a floating point using a width of 6 and 2 decimal digits.

```
The average of 100 and 90 is  95.00
```

%d

%6.2f

Common printf Formats

Format	Meaning	Examples
%d	decimal integers	`%d  %6d`
%f	fixed pt. floating-point	`%f %10.4f`
%e	scientific notation	`%10e  (1.2345e-02)`
%g	general floating point	`%10g`
	(use %e or %f, whichever is more compact)	
%s	String	`%s  %10s  %-10s`
%c	Character	`%c`

```
String owner = "Taksin Shinawat";
int acctNumber = 12345;
double balance = 4000000000;
System.out.printf("Account %6d Owner %-18s has %10.2f\n",
        acctNumber, owner, balance );
```

```
Account  12345 Owner Taksin Shinawat   has  4000000000.00
```

More details on printf

- **`printf`** is an *instance of the* **Formatter** class.
- It is predefined for **`System.out`**.
- System.out.**printf**(...) is same as System.out.**format**(...).
- For complete details see Java API for "Format".
- For tutorial, examples, etc. see:
  - Sun's *Java Tutorial*
  - *Core Java*, page 61.

String.format

A useful method that creates a String using a format.

```
/** describe a bank acount */
String name = "John Doe";
long accountId = 12345;
double balance = 123.456;
String result = String.format(
 "Acct: %09d  Owner: %s Balance: %.2f",
        accountId, name, balance );
return result;
```

```
Acct: 000012345  Owner: John Doe Balance: 123.46
```

Input

Input byte-by-byte

`System.in` is an InputStream object.

It reads data one byte at a time, or an array of bytes.

Use System.in.read( ) to get "raw" data, such as an image:

```
int a = System.in.read( );
if (a < 0) /* end of input */;
else {
  byte b = (byte)a;
  handleInput( b );
}
```

Boring, isn't it?

Input Line-by-Line

To get a line of input as a String, you can create a BufferedReader object that "wraps" System.in.

```
BufferedReader reader = new BufferedReader(
   new InputStreamReader( System.in ) );
String s = reader.readLine( ); // read one line
```

The readLine( ) method removes the NEWLINE (\n) from the input, so you won't see it as part of the string.

Check for end of data

If there is no more data in the input stream, readLine( ) returns a null String.

For console input, readLine( ) will *wait* (*block*) until user inputs something.

Here is how to test for end of data:

```
BufferedReader reader = new BufferedReader(
   new InputStreamReader( System.in ) );
String s = reader.readLine( );  // read one line
if ( s == null ) return;        // end of data
```

Handling I/O Errors

When you use System.in.read or a BufferedReader an input error can occur -- called an *IOException*.

Java requires that your program either "catch" this exception to declare that it might "throw" this exception.

To be lazy and "throw" the exception use:

```
public static void main(String [] args)
                        throws IOException {

   BufferedReader reader = new BufferedReader(
      new InputStreamReader( System.in ) );
   // read a line of input
   String inline = reader.readLine( );
```

Catching I/O Errors

To "catch" the exception and do something, use:

```
BufferedReader reader = new BufferedReader(
          new InputStreamReader( System.in ) );

// read a line of input.
// display message and return if error
String line = null;
try {
    line = bin.readLine( );
     buf.append( line );
}
catch( IOException ioe ) {
   System.err.println( ioe.getMessage() );
   return;
}
```

Flexible Input: Scanner class

The Scanner class allow much more flexible input.

A Scanner can:

- read entire line or one word at a time
- test for more data
- test if the *next* input (word) is an int, long, double, etc.
- read input and convert to int, long, float, double
- *skip unwanted data*
- report errors (*Exceptions*) that occur

Import Scanner

Scanner is a "utility" so it is package **`java.util`**.

You should import this class to use it:

```
package myapp;
import java.util.Scanner;
...

public class InputDemo {
```

Create a Scanner Object

Scanner "wraps" an InputStream.

You give the InputStream object as parameter when you create a Scanner object...

```
// create a Scanner to read from System.in
Scanner console = new Scanner( System.in );
```

Where to Create Scanner object?

1) You can create a Scanner as a local variable

```
public void myMethod( ) { // no IOException !
   // create a Scanner as a local variable
   Scanner in = new Scanner( System.in );
   // read some different types of data
   int count = in.nextInt( );
```

2) or create as an attribute.
Typically a *static* attribute since System.in is static.

```
public class InputDemo {
   // create a Scanner as static attribute
   static Scanner console =
        new Scanner( System.in );

   // can use console in any method.
```

Using Scanner

Look at some simple examples

```
public void myMethod( ) { // no IOException !
   // create a Scanner to process System.in
   Scanner in = new Scanner( System.in );

   // read some different types of data
   int count = in.nextInt( );
   long big  = in.nextLong( );
   float x   = in.nextFloat( );
   double y  = in.nextDouble( );

   // read Strings
   String word = scan.next( ); // next word
   String line = scan.nextLine( ); // next line
```

Input Errors

If you try to read an "int" but the next input is *not* an integer then Scanner throws an InputMismatchException

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

// read a number
System.out.print("How many Baht? ");
int amount = scan.nextInt( );
```

convert next input word to an "int"

```
How many Baht?   I don't know
Exception in thread "main"
java.util.InputMismatchException
```

How to Test the Input

Scanner has methods to test the next input value:

```
Scanner scanner = new Scanner( System.in );
int amount;
// read a number
System.out.print("How many Baht? ");
if ( scanner.hasNextInt( ) )
   amount = scanner.nextInt( );
else {
   System.out.println("Please input an int");
   scanner.nextLine(); // discard old input
}
```

true if the next input value is an "int"

```
How many Baht?   I don't know
Please input an int
```

Testing the input

So now you can check for invalid input.

But, what do you do with the bad input?

If scan.hasNextInt( ) is false, then the program doesn't read it.

So, the bad input is still waiting to be read. Like this:

```
How many Baht?    I don't know
```

Next input value to read.

We want to remove this bad input, so we can ask the user to try again. ...what should we do?

Discarding the input

If the input is wrong, then *throw it away* by reading the line and discarding it.

```
Scanner scanner = new Scanner( System.in );
int amount;
// read a number
System.out.print("How many Baht? ");
if ( scanner.hasNextInt( ) )
   amount = scanner.nextInt( );
else {
   System.out.println("Please input an int");
   scanner.nextLine(); // discard input
```

get the input line but don't assign it to anything! (discard it)

Useful Scanner Methods

Return type	Method	Meaning
String	`next()`	get next "word"
String	`nextLine()`	get next line
int	`nextInt()`	get next word as int
long	`nextLong()`	get next word as long
float	`nextFloat()`	get next word as float
double	`nextDouble()`	get next word as double
boolean	`hasNext()`	true if there is more input
boolean	`hasNextInt()`	true if next word can be "int"
boolean	`hasNextLong()`	true if next word can be "long"
boolean	`hasNextFloat()`	true if next word can be "float"
boolean	`hasNextDouble()`	true if next word can be "double"

See Java API for java.io.Scanner for a complete list of methods.

Input/Output Example

Read some numbers and output their sum...

```
Scanner scanner = new Scanner( System.in );
double sum = 0.0;
// prompt user for input
System.out.print("Input some numbers: ");
// read as long as there are more numbers
while ( scanner.hasNextDouble( ) ) {
   double x = scanner.nextDouble( );
   sum = sum + x;
}
System.out.println();
System.out.println("The sum is "+sum);
```