Output (Ch. 6.6.1)

- **Info we get from the user is <u>input</u>**
  - Either from the keyboard, mouse, or something else like a network connection or other type of cable

- **Info we show to the user is <u>output</u>**
  - This can be a message, additional information, even a request that the user perform some kind of action. Graphics and noises are also output, but here we will deal with information printed to the console.
  - These messages and information are typically "hard-coded" into the program itself, then pushed out to the screen, console, or monitor using specific print-to-the-screen mechanisms (services) that are a part of Java

Java Console Window

The console window that starts a Java application is typically known as the *standard output* device.

The *standard input* device is typically the keyboard.

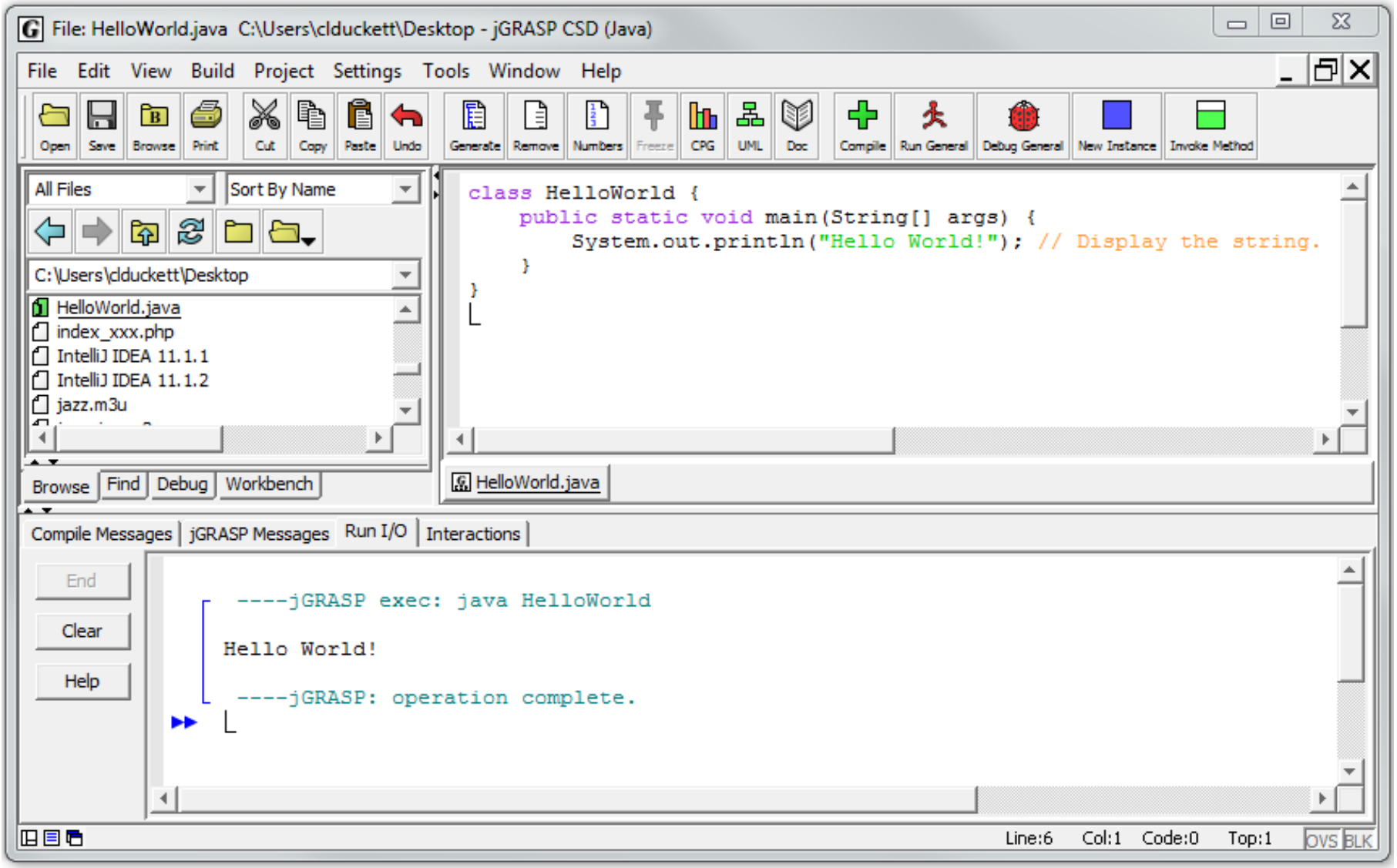
Java sends information to the standard output device by using a Java class stored in the standard Java library.

In **jGRASP**, this output is displayed as part of Compile Messages

Java classes in the standard Java library are accessed using the **Java Applications Programming Interface** (API).

The standard Java library is commonly referred to as the **Java API**. You do not need to add special code to access the standard library.

Hello, World!


File: HelloWorld.java C:\Users\clduckett\Desktop - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
Open Save Browse Print Cut Copy Paste Undo Generate Remove Numbers Freeze CPG UML Doc Compile Run General Debug General New Instance Invoke Method
All Files
Sort By Name
C:\Users\clduckett\Desktop
HelloWorld.java
index_xxx.php
IntelliJ IDEA 11.1.1
IntelliJ IDEA 11.1.2
jazz.m3u
Browse Find Debug Workbench
class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello World!"); // Display the string.
}
}
HelloWorld.java
Compile Messages jGRASP Messages Run I/O Interactions
End
Clear
Help
----jGRASP exec: java HelloWorld
Hello World!
----jGRASP: operation complete.
Line:6 Col:1 Code:0 Top:1 OVS BLK

Printing to the Console

Example:

```
System.out.println("Programming is great fun!");
```

This line uses the **System** class from the standard Java library.

The **System** class contains methods and objects that perform system level tasks.

The **out** object, a member of the **System** class, contains the methods **print** and **println**.

'println' => Print Line

The **print** and **println** methods actually perform the task of sending characters to the output device.

The line:

```
System.out.println("Programming is great fun!");
```

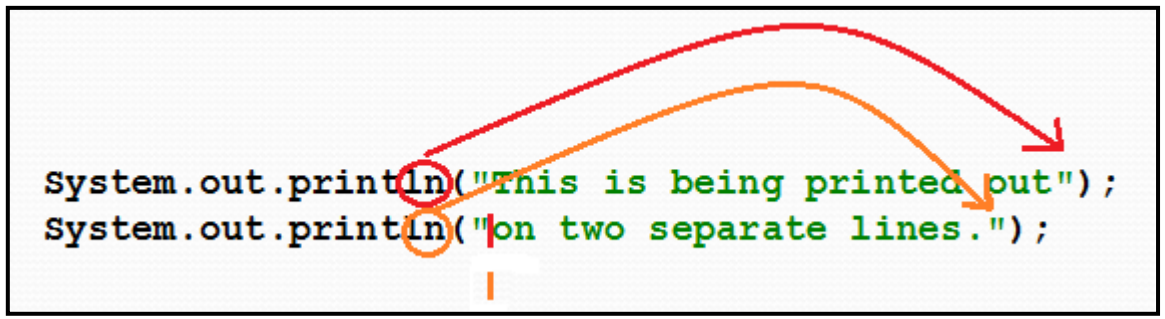
is pronounced: *System **dot** out **dot** printline*

The value inside the parenthesis will be sent to the output device (in this case, a **string**).

The **println** method places a newline character at the **end** of whatever is being printed out.

The following lines would be printed out on separate lines since the first statement sends a newline command to the screen:

```
System.out.println("This is being printed out");
System.out.println("on two separate lines.");
```



'print'

The **`print`** statement works very similarly to the **`println`** statement.

However, the **`print`** statement does not put a newline character at the end of the output.

The lines:


```
System.out.print("These lines will be");
System.out.print("printed on");
System.out.println("the same line.");
```

Will output:

```
These lines will beprinted onthe same line.
```

Add structure to your strings

For all of the previous examples, we have been printing out strings of characters.

Later, we will see that much more can be printed.

There are some special characters that can be put into the output.

```
System.out.print("This line will have a newline at
the end.\n");
System.out.print("This doesn't show up on the same
line.");
```

The **\n** in the string is an **escape sequence** that represents the newline character.

Escape sequences allow the programmer to **print** characters that otherwise would be **unprintable**.

Java Escape Sequences – Special Characters

`\n`	newline	Advances the cursor to the next line for subsequent printing
`\t`	tab	Causes the cursor to skip over to the next tab stop
`\b`	backspace	Causes the cursor to back up, or move left, one position
`\r`	carriage return	Causes the cursor to go to the beginning of the current line, not the next line
`\\`	backslash	Causes a backslash to be printed
`\'`	single quote	Causes a single quotation mark to be printed
`\"`	double quote	Causes a double quotation mark to be printed

Java Escape Sequences – Special Characters

- Even though the escape sequences are comprised of two characters, they are treated by the compiler as a single character.

```
System.out.print("These are our top sellers:\n");
System.out.print("==========================\n");
System.out.print("\t- Computer games\n\t- Coffee\n ");
System.out.println("\t- Aspirin");
```

Would result in the following output:

```
These are our top sellers:
 ==========================
    - Computer games
    - Coffee
    - Aspirin
```

- With these escape sequences, complex text output can be achieved.

The '+' Character

The + operator can be used in two ways.
(1) as a **concatenation** operator
(2) as an **addition** operator

If either side of the **+** operator is a string, the result will be a string.

```
System.out.println("Hello " + "World!");
                   Hello World!
System.out.println("The value is: " + 5);
                   The value is: 5
System.out.println("The value is: " + value);
                   The value is: <what value is>
System.out.println("The value is" + ": \n" + 5);
                   The value is:
                   5
```

Concatenation

- Java commands that have **string literals** must be treated with care.
- A string literal value **cannot** span lines in a Java source code file.

```
System.out.println("This line is too long and now it
has spanned more than one line, which will cause a
syntax error to be generated by the compiler. ");
```

- The **String concatenation operator can** be used to fix this problem.

```
System.out.println("These lines are " +
                   "now ok and will not " +
                   "cause the error as before.");
```

- String concatenation can join various data types, too.

```
System.out.println("We can join a string to " +
                   "a number like this: " + 5);
```

More fun with concatenation

- The **concatenation operator** can be used to format complex **String** objects.

```
System.out.println("The following will be printed " +
                   "in a tabbed format: " +
                   "\n\tFirst = " + 5 * 6 + ", " +
                   "\n\tSecond = " + (6 + 4) + "," +
                   "\n\tThird = " + 16.7 + ".");
```

- Notice that if an **addition operation** is also needed, it must be put in parenthesis.

```
The following will be printed in a tabbed format:
        First = 30,
        Second = 10,
        Third = 16.7.
```

Notes on non-robotic programming

```
public class nonRobotClass extends Object
{

  public static void main(String[] args)
  {
     System.out.println( “I printed!");  // but, do something more interesting here.
  }
}
```

- You don't need becker.jar, because that is included to define the robot-based classes.
- import java.util.*; ← You may include other classes, like this util package
- This new class extends 'Object' because 'Object' is the most basic class and we are starting basic.

Getting started without robots

```
public class nonRobotClass extends Object
{

  public static void main(String[] args)
  {
     System.out.println( “I printed!");  // but, do something more interesting here.
  }
}
```

- You don't need becker.jar, because that is included to define the robot-based classes.
- import java.util.*; ← You will probably include other classes, like this util package
- This new class extends 'Object' because 'Object' is the most basic class and we are starting basic.

Using output with Robots

`String | toString() This robot, represented as a string.`

```
12      Robot aRobot = new Robot(theCity, 3, 0, Direction.EAST, 20);
13      System.out.println (aRobot);
```

```
becker.robots.Robot[street=3, avenue=0, direction=EAST,
isBroken=false, numThingsInBackpack=20]
```