Honors Computer Programming 1-2

Introduction To Chapter 1

CHAPTER GOALS

- To understand the activity of programming
- To learn about the architecture of computers
- To learn about machine code and high level programming languages
- To become familiar with your computing environment and your compiler
- To compile and run your first Java program
- To recognize syntax and logic errors

What is a computer?

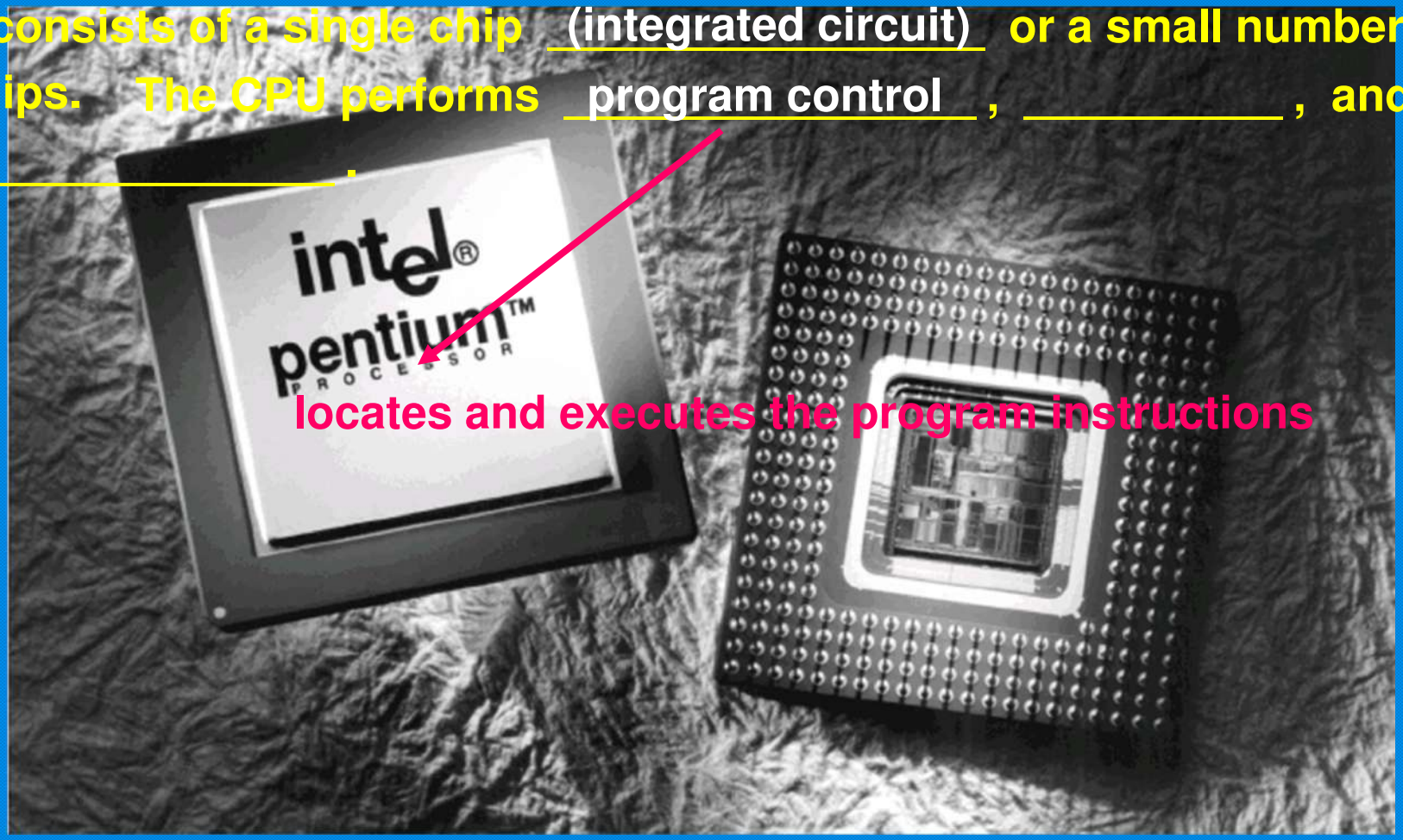
A computer must be __programmed__ to perform tasks. A computer itself is a machine that __stores data__ (numbers, words, pictures) and executes __programs__. Programs are sequences of __instructions and decisions__ that the computer carries out to achieve a task.

Programmers develop __computer programs__ to make computers perform new tasks. The art of designing and implementing these programs is called __computer programming__. To __use__ a computer, you do not need to do any programming. In this course we will begin the foundation for the career of __computer scientist__.

The Anatomy of a Computer?

The heart of a computer is the central processing unit (CPU) .

It consists of a single chip (integrated circuit) or a small number of chips. The CPU performs program control , _____ , and _____ .



locates and executes the program instructions

The Anatomy of a Computer?

The heart of a computer is the central processing unit (CPU) .

It consists of a single chip (integrated circuit) or a small number of chips. The CPU performs program control , arithmetic , and _____ .

carries out arithmetic operations such as addition, subtraction, multiplication, and division

The Anatomy of a Computer?

The heart of a computer is the central processing unit (CPU) .

It consists of a single chip (integrated circuit) or a small number of chips. The CPU performs program control , arithmetic , and data movement .

fetches data from external memory or devices. All data must travel through the CPU.

The Anatomy of a Computer?

The heart of a computer is the central processing unit (CPU) .

It consists of a single chip (integrated circuit) or a small number of chips. The CPU performs program control , arithmetic , and data movement .

The computer keeps data in storage . There are two kinds of storage:

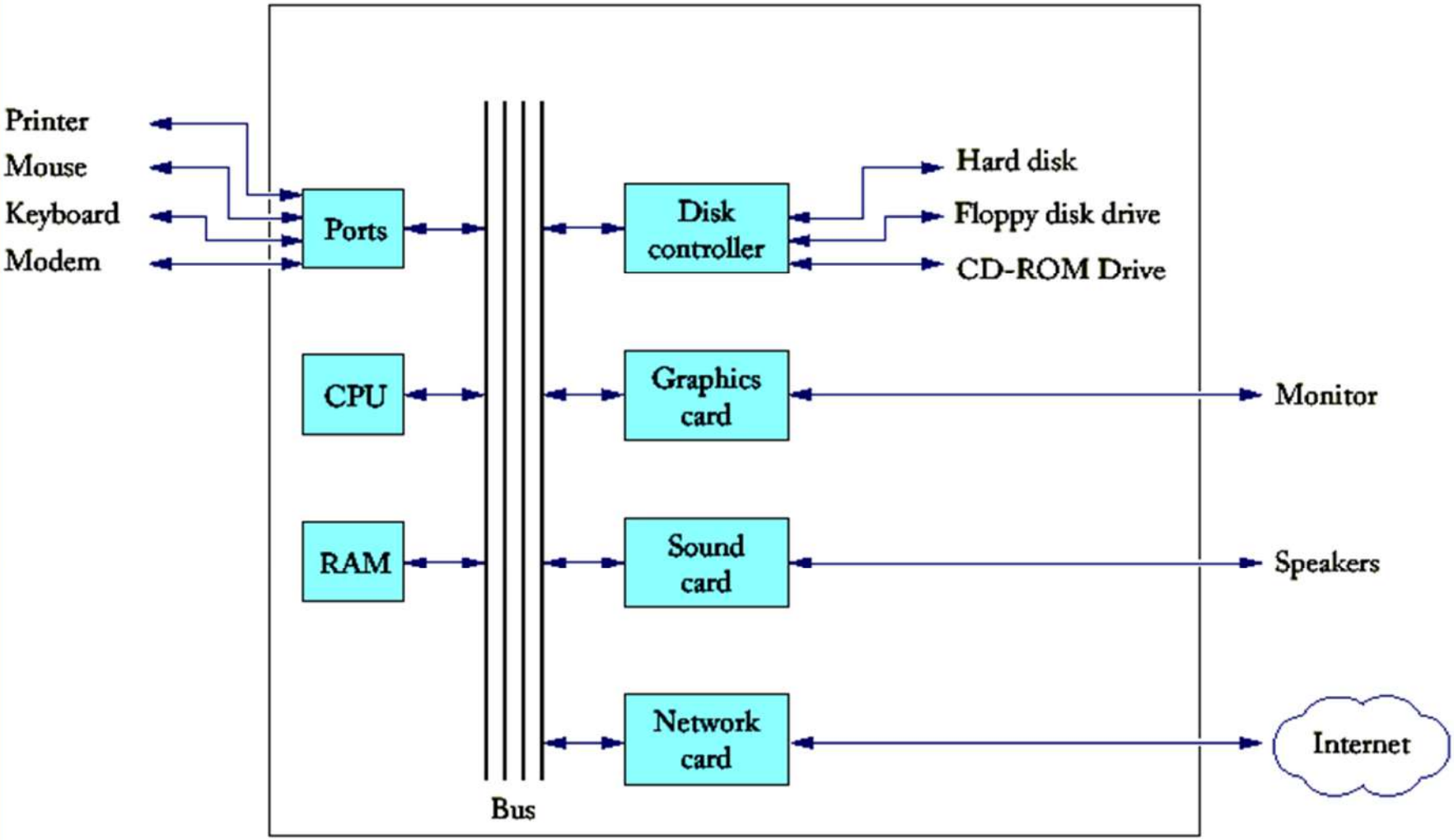
- Primary storage is also called random-access memory or RAM . Primary storage is made of memory chips .
  It loses all its data when the power is turned off .
- Secondary storage is usually a hard disk . A hard disk consists of rotating platters .

The Anatomy of a Computer?

Some computers are self-contained units while others are interconnected through networks . Most computers have removable storage devices such as floppy disks , tapes , or compact disks (CDs) . To interact with a human user a computer requires peripheral devices such as monitor , speakers , and printers .

The CPU, the RAM, and the electronics controlling the hard disk and other devices are interconnected through a set of electrical lines called a bus . Data travel along the bus from the RAM and peripheral devices to the CPU and back. A motherboard contains the CPU, the RAM, and card slots through which cards that control peripheral devices connect to the bus .

Schematic Diagram of a Computer



Programming Languages

The CPU executes machine instructions . CPUs from different vendors such as Intel Pentium, Sun Solaris, or Macintosh OSX have different sets of machine instructions. To enable Java to run on multiple CPUs, Java contains machine instructions for a "Java Virtual Machine" . This is an idealized CPU that is simulated by a program run on the actual CPU .

A typical sequence of machine instructions on a Java Virtual Machine or JVM is:

```
21  40  16  100  163  240
```

Note that this machine code is encoded as numbers so that they may be stored in memory.

machine code: `21 40` `16 100` `163 240`

These instructions mean:

- load the contents of memory location 40
- load the value 100
- if the first value is greater than the second value continue with the instruction stored in memory location 240

machine code: `21 40` `16 100` `163 240`

This machine code was used by early computers but was difficult to program.

The box at the right shows the same code but with "commands" stored as numbers.
For example, the command "iload" denotes integer load and is translated by an assembler into 21.

Assembly Code

```
iload      40
bipush     100
if_icmpgt  240
```

And the command "bipush" denotes push integer constant and is translated by an assembler into 16. And the command "if_icmpgt" denotes if integers compare greater and is translated by an assembler into 163. The assembly language is a big improvement over machine code since it is easier to read.

In the mid 1950s, high-level programming languages began to appear. These high-level languages were easier to read but needed a program called a compiler to translate programs into machine code. In this course we will use a high-level language called Java.

Java Code

```
if (intRate > 100)
        System.out.print ("Interest rate error");
```

is equivalent to

Assembly Code

```
iload   40
bipush  100
if_icmpgt  240
```

is equivalent to

Machine Code

```
21   40   16   100   163   240
```

The Java Programming Language

In 1991, programmers at Sun Microsystems began development of a high-level programming language called Java . The language was designed to be simple and architecture-neutral meaning that it will run on a variety of hardware . Since 1996, browsers such as Netscape and Internet Explorer have been able to download programs called applets and run them. Applets are written in Java .

Applets on a Web Page

Jmol - Netscape 6

File Edit View Search Go Bookmarks Tasks Help

http://jmol.sourceforge.net/index.html?right=JmolApplet.html

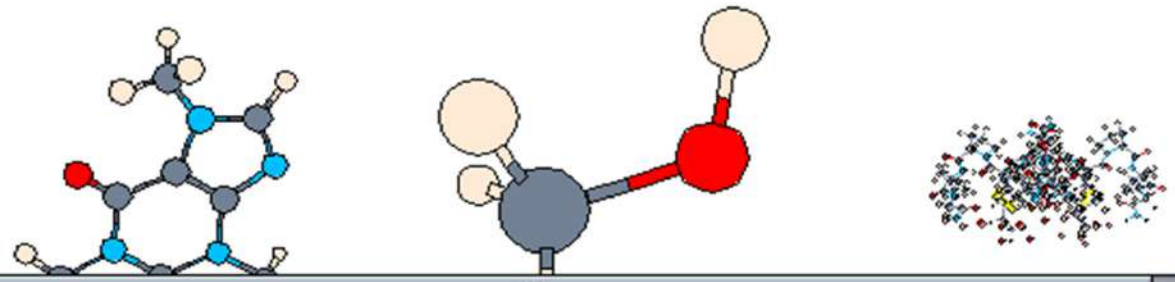
Go Search

Main Page
Features
FAQ
Screen Shots
Jmol Applet
Download
Mailing Lists
News
Contributors
Bugs

Jmol

Jmol Applet

Tom Grey (t.grey@ic.ac.uk), Bradley A. Smith (bradley@baysmith.com), and Matthew Fagan (mfagan@physics.usyd.edu.au) have done some amazing work converting Jmol for use as an Applet. Here are three samples of the Jmol Applet with Caffeine (XYZ), Methanol (CML), and deamino–oxytocin (PDB):



Keys: S– change style; L– Show labels; B–Toggle Bonds

The Java Programming Language

Java has a rich library making programs portable . That is Java programs will run without change on Windows, UNIX, Linux, or the Macintosh. Java provides a high degree of safety since it has an assortment of security features that guarantee that no evil applet can run on your computer.

Because Java was not specifically designed for students , no thought was given to make it simple to write basic programs . Furthermore, you cannot expect to learn all of Java in one semester . The central goal of this book is not to make you memorize Java but rather to teach you how to think about programming.

Compiling a Simple Program

The traditional first program in a new language is a program that displays a simple greeting: `Hello, World!` .

The code is shown in the box below.

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

Java is case sensitive . You cannot type `MAIN` or `printLn` .

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

Java has free-form layout .

You could type the same code as shown below.

```
public class Hello{public static void main(String[ ]
args){// display a greeting in the console window
System.out.println ("Hello, World!");}}
```

But don't -- good taste dictates that you lay out your programs so that they are easy to read .

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

The first line starts a new `class`. In Java, every program consists of one or more classes. The keyword `public` denotes that the class is usable by the "public". You will later encounter `private` features which are not usable by the public.

At this time regard the structure shown at the right as a necessary part of the "plumbing" required to write any Java program.

```
public class ClassName
{
   ...
}
```

```
public class Hello
{
    public static void main(String[ ] args)
    {
        // display a greeting in the console window
        System.out.println ("Hello, World!");
    }
}
```

name this file Hello.java

In Java, every source file can contain at most one public class and the name of the public class must match the name of the file. For example, the class `Hello` must be contained in a file named `Hello.java`. Do not name the class `HELLO` and the file `hello.java`. It is important that the names and the capitalization match exactly.

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

The construction

```
public static void main(String[ ] args)
{
   ...
}
```

defines a method called `main`. A method contains a collection of programming instructions. Every Java application must have a `main` method. Most Java programs contain other methods besides `main`.

```
public class Hello
{
    public static void main(String[ ] args)
    {
        // display a greeting in the console window
        System.out.println ("Hello, World!");
    }
}
```

note that the parameter is enclosed within parenthesis

The parameter `String[ ] args` is a required part of the `main` method. The keyword `static` indicates that the main method does not operate on an object. Most methods do operate on an object so `static` methods are not common. Nevertheless the `main` method must be `static`.

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

At this time consider the code shown at the right another part of the "plumbing".

```
public class Hello
{
   public static void main(String[ ] args)
   {
      ...
   }
}
```

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

The first line inside the main method is a comment . Comments are used to explain the program to other programmers or to yourself. Any text between the // and the end of the line are ignored by the compiler.

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

The instructions for the `main` method, that is the statements between curly braces `{ }` are executed one by one. Each statement ends in a semicolon `;` . Our `main` method has a single statement `System.out.println("Hello, World!")`. This statement prints a line of text namely `Hello, World!` .

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

You need to specify where to send that string: to a window, to a file, to a networked computer, etc. In this case, the destination is the console window represented in Java by an object called `out`. The designers of Java placed the `out` object in the `System` class. To use the out object of the `System` class you must refer to it as `System.out`.

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

To use an object like `System.out`, you specify what you want to do with it. In this case you want to print a line of text. The `println` method carries out this task. Whenever you want to call a method in Java you need to specify three things:

- the object you want to use -- in this case `System.out`
- the name of the method you want to use -- in this case `println`
- a pair of parenthesis containing any other information the method needs -- in this case `("Hello, World!")`

```
public class Hello
{
   public static void main(String[ ] args)
   {
      // display a greeting in the console window
      System.out.println ("Hello, World!");
   }
}
```

Note the two periods in `System.out.println` have different meanings. The first period means "locate the `out` object in the `System` class" while the second period means "apply the `println` method to the `System.out` object".

Method Calls

syntax → `object.methodName(parameters)`

example → `System.out.println("Hi Mr. A");`

purpose → to invoke a method on an object and supply any additional parameters

A sequence of characters enclosed in quotation marks such as `"Hello, World!"` is called a string . It is an error to omit either quotation mark.

You can also print numerical values. For example, the statement

```
System.out.println(3 + 4)
```

will display 7 .

The `println` method prints a string or a number and then starts a new line .

The statements

```
System.out.println("Hello");
System.out.println("World!");
```

will display

```
Hello
World!
```

There is another method called `print` that you can use to print an item without starting a new line afterward.

The statements

```
System.out.print("00");
System.out.println(3 + 4);
```

will display

```
007
```

Omitting Semicolons

In Java, every statement must end in a semicolon . Forgetting the semicolon causes a compiler error.

The statements

```
System.out.println("Hello")
System.out.println("World!");
```

causes a compiler error since the first line does not end in a semicolon .

Alternative Syntax for Comments

You have already learned that the compiler ignores anything you type between __//__ and the __end of the line__. The compiler also ignores anything you type between __/*__ and __*/__. For example, __/* A simple Java program */__.

The __// comment__ is easier if the comment is only a single line. If you have a comment that is longer than one line, then the __/* .... */__ style is preferred.

Use the style shown below for multiple-line comments.

```
/**
*  This is a simple Java program that you can
*  use to try out your compiler and interpreter
*/
```

Escape Sequences

Suppose you want to display a string containing quotation marks such as `Hello, "World"!` . To print the quotation marks you will need to use the backslash `\` character. Inside a string, the sequence `\"` denotes a literal quote. The correct display statement is `System.out.println("Hello, \"World\"!");` .

The backslash character `\` is an escape character and the character sequence `\"` is an escape sequence . The backslash does not denote itself; instead it is used to encode other characters that would otherwise be difficult to include in a string.

To print a backslash itself, you must enter \\ within a string.

The statement:

```
System.out.println("The secret message is in C:\\Temp\\Secret.txt");
```

will output The secret message is in C:\Temp\Secret.txt .

Another escape sequence occasionally used is \n which denotes a newline or line feed character. The statement

```
System.out.print("*\n**\n***\n");
```

prints the output

```
*
**
***
```

Finally, escape sequences are useful for including international characters in a string. For example, suppose that you want to print `"All the way to San José!"`, with the accented letter é.

Java uses the Unicode encoding scheme to denote international characters. For example, the character é has Unicode encoding `00E9`. The correct output is

```
System.out.print("All the way to San Jos\u00E9!")
```

Additional codes can be looked up in appendix A6.

Errors

Consider the code shown at the right.

```
System.ouch.print("Hello, World!");
System.out.print("Hello, World!);
System.out.print("Helo, World!");
```

- In the first line, a compiler error will be generated. That is because there is no ouch object in the System class. A possible compiler error message might be undefined symbol ouch . When a compiler error occurs the program will not run and you must fix the error.

- In the second line, a compiler error will be generated. This compiler error is usually referred to as a syntax error due to missing quote mark at the end of the string. The error message might be " expected . Again you must fix the error before the program will run .

Errors

```
System.ouch.print("Hello, World!");
System.out.print("Hello, World!);
System.out.print("Helo, World!");
```

- The error in the third line is different. The program will compile and run but the output is wrong. The third line will print Helo, World! . This error is not a compiler error since the program runs . It is called a run-time error or a logic error. A run-time or logic error occurs when the program runs but doesn't do what it's supposed to do .

Errors

```
public class Hello
{
  public static void Main(String[ ] args)
  {
    System.out.println ("Hello, World!");
  }
}
```

Consider the code shown.
This contains a compiler error since the program will not compile. This is due to the spelling error in the third line: `Main` should be changed to `main`. The compiler will not recognize `Main` is `main` and will generate the error.

In general, compiler errors are detected by the compiler but logic errors are detected (hopefully) by testing the program.

The Compilation Process

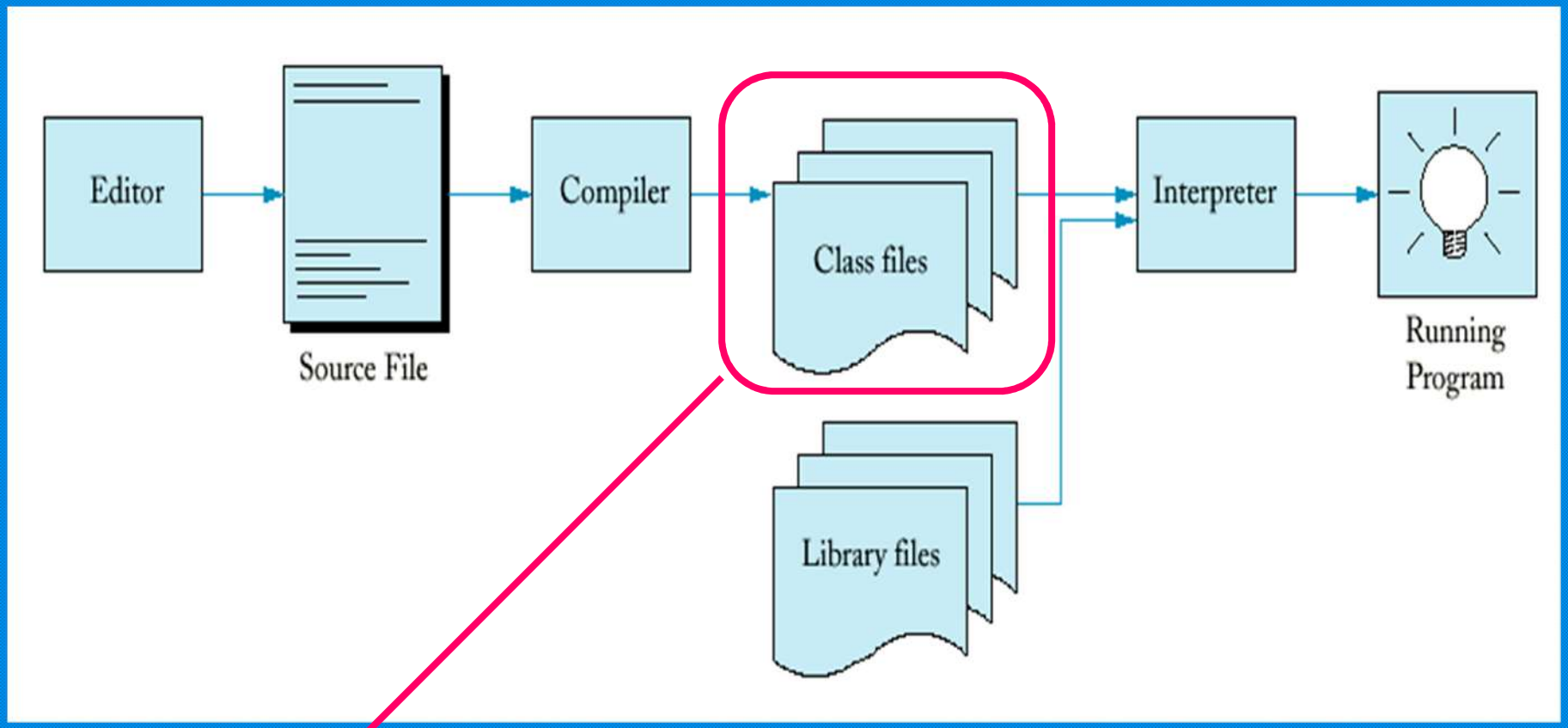
The compilation process involves several components. They are:

- editor → a program for entering and modifying text such as a Java program
- source file → file containing statements you wrote using the editor having an extension of `.java`
- compiler → a program that translates the source file into bytecode which consists of virtual machine instructions. The bytecode for Hello.java is stored in a file with the name `Hello.class`
- libraries → a collection of code that has been programmed by someone else ready for you to use in your program. The information in your source code is not enough to make a running program -- you need the code from the libraries .

The Compilation Process

- **interpreter** → the Java interpreter loads the __bytecode__ of the program you wrote, __starts__ your program, and __loads__ the necessary __library__ files as they are required.

From Source Code to Running Program



bytecode

The Edit-Compile-Test Loop

You start in the __editor__ writing the __source__ file. You __compile__ the program and look for __error__ messages. You go back to the editor and fix any __syntax__ errors. When the compiler succeeds, you __run__ the program. If you find any __run-time__ errors, you will need to go back to the __editor__ and fix them. You __compile__ again to see if the error has gone away. If not, go back to the __editor__ .

also called compiler errors

also called logic errors