

Topic 1:
Introduction to OOP Programming
Using Java

What is a Program?

- Program is a set of instructions written in a programming language (i.e high level Language such as Java) that is supposed to perform some specific tasks.
- Very Basic Program Construction include:
 - Sequential
 - Selection
 - Iteration
 - I/O (input and output)
 - Arrays (Data Structure)

The Java Programming Language

- Simple
- Safe
- Platform-independent ("write once, run anywhere")
- Rich library
- Designed for the internet

The Java Programming Language

- Java
 - Based on C and C++
 - Originally developed in early 1991 for intelligent consumer electronic devices
 - Market did not develop, project in danger of being cancelled
 - Internet exploded in 1993, saved project
 - Used Java to create web pages with dynamic content
 - Java formally announced in 1995
 - Now used to create web pages with interactive content, enhance web servers, applications for consumer devices (pagers, cell phones)...

The Java Programming Language

- Java programs
 - Consist of pieces called classes
 - Classes contain methods, which perform tasks
- Class libraries
 - Also known as Java API (Applications Programming Interface)
 - Rich collection of predefined classes, which you can use
- Two parts to learning Java
 - Learning the language itself, so you can create your own classes
 - Learning how to use the existing classes in the libraries

Basics of a Typical Java Environment

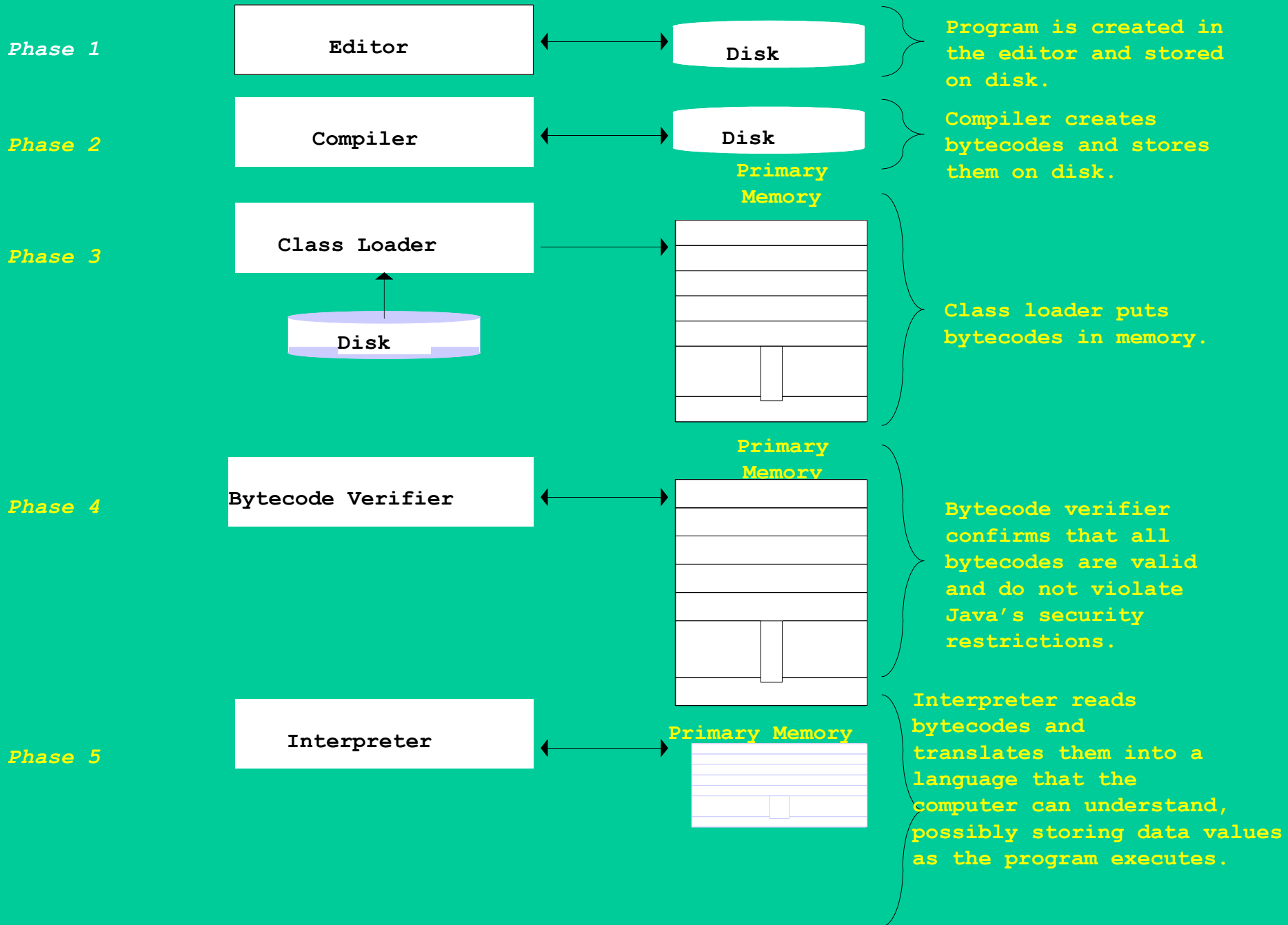
- Java Systems
 - Consist of environment, language, Java Applications Programming Interface (API), Class libraries
- Java programs have five phases
 - Edit
 - Use an editor to type Java program
 - **vi** or **emacs**, notepad, Jbuilder, Visual J++
 - **.java** extension
 - Compile
 - Translates program into bytecodes, understood by Java interpreter
 - **javac** command: **javac myProgram.java**
 - Creates **.class** file, containing bytecodes (**myProgram.class**)

Basics of a Typical Java Environment

- Java programs have five phases
 - Loading
 - Class loader transfers **.class** file into memory
 - Applications - run on user's machine
 - Applets - loaded into Web browser, temporary
 - Classes loaded and executed by interpreter with **java** command
 - java Welcome**
 - HTML documents can refer to Java Applets, which are loaded into web browsers. To load,
 - appletviewer Welcome.html**
 - **appletviewer** is a minimal browser, can only interpret applets

Basics of a Typical Java Environment

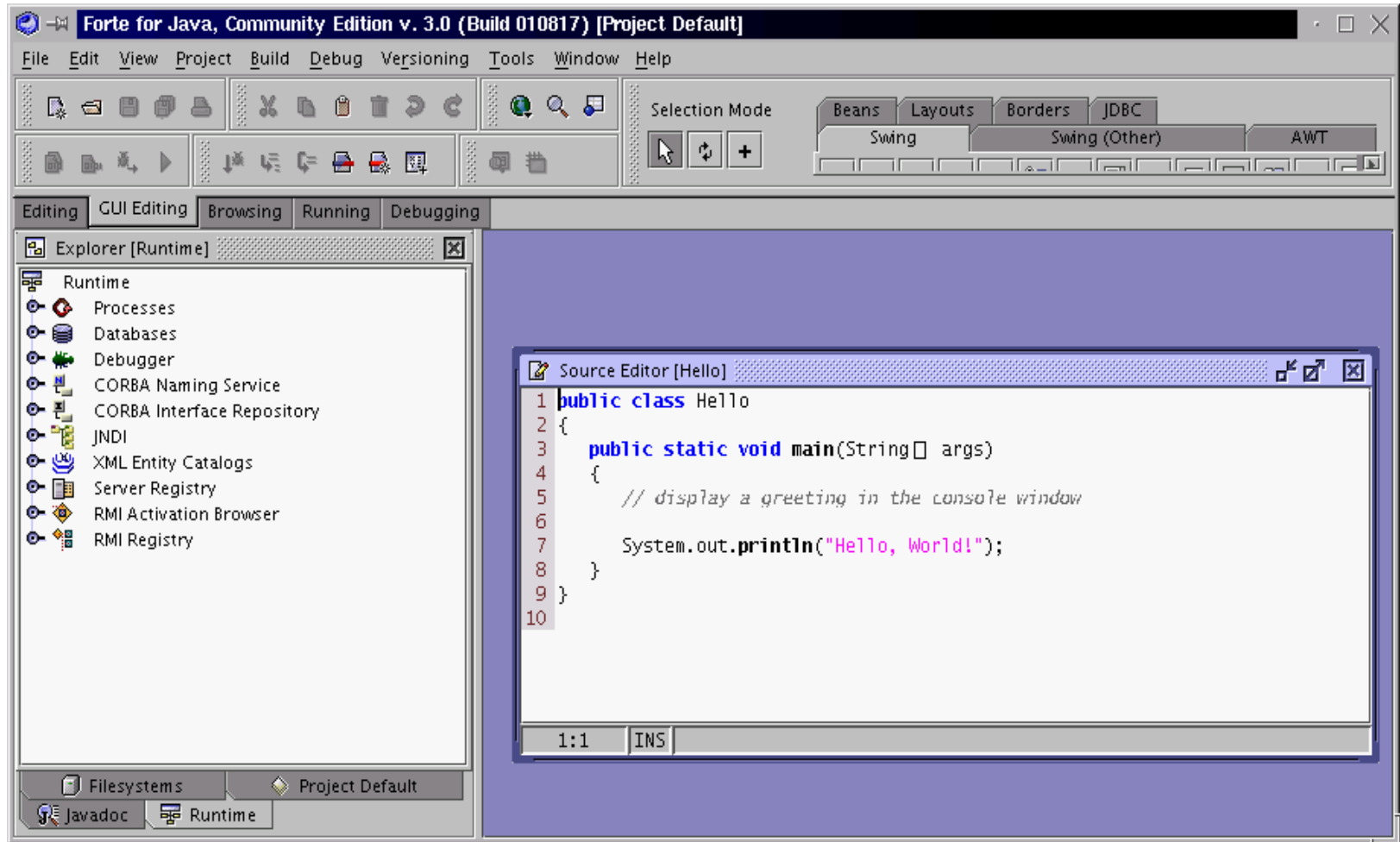
- Java programs have five phases
 - Verify
 - Bytecode verifier makes sure bytecodes are valid and do not violate security
 - Java must be secure - Java programs transferred over networks, possible to damage files (viruses)
 - Execute
 - Computer (controlled by CPU) interprets program one bytecode at a time
 - Performs actions specified in program
 - Program may not work on first try
 - Make changes in edit phase and repeat



General Notes about Java

- Just-in-time compiler
 - Midway between compiling and interpreting
 - As interpreter runs, compiles code and executes it
 - Not as efficient as full compilers
 - Being developed for Java
 - Integrated Development Environment (IDE)
 - Tools to support software development
 - Several Java IDE's are as powerful as C / C++ IDE's

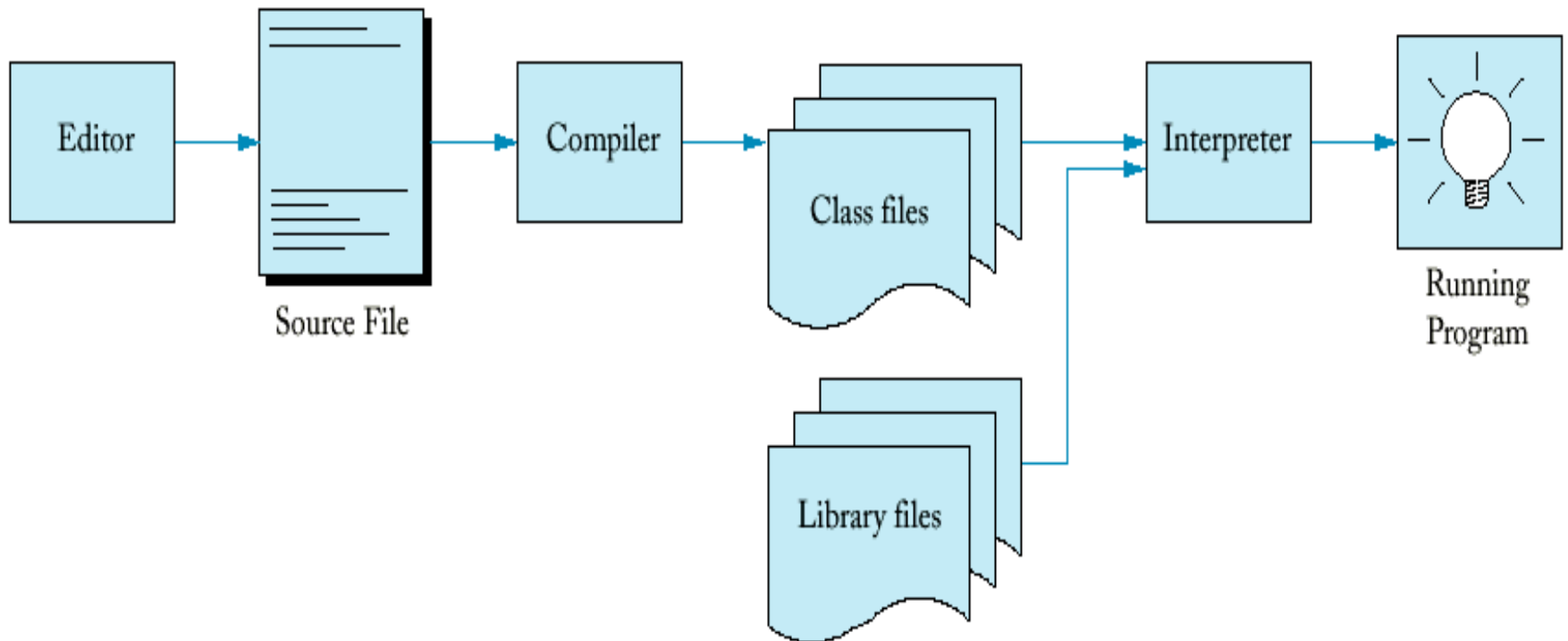
An Integrated Development Environment



Compiling and Running

- Type program into text editor
- Save
- Open command shell
- Compile into byte codes
`javac Hello.java`
- Execute byte codes
`java Hello`

From Source Code to Running Program



Syntax Errors vs. Logic Errors

- Syntax errors

```
System.ouch.print ("...");  
System.out.print ("Hello");
```

- Detected by the compiler

- Logic errors

```
System.out.print ("Helo");
```

- Detected (hopefully) through testing

Simple Basic Java

- Plain Vanilla HelloWorld •
Style Program:
 - **Example 1:**
 1. `/* our first program!!!!*/`
 2. `public class Prog1`
 3. `{`
 4. `public static void main (String args[])`
 5. `{`
 6. `System.out.println("Dopey");`
 7. `System.out.println("Grumpy");`
 8. `}`
 9. `}`
- **Let's look at the program line by line:**
 - **Line 1** is a comment. As in C++, comments begin with the symbol `//` and continue to the end of the line. Multiple line comments are enclosed by the symbols `/*` and `*/`.
 - **(Line 2)**
 - The word *class* indicates that we are defining a new class. The name of the class is *Prog1*. (A class is a blueprint for objects, so, in theory, we could create a Prog1 object from class Prog1).
 - The keyword *public* is an **access modifier**. A public class is accessible anywhere in a program. We will discuss this in greater detail later. C++ does not have access modifiers for classes.

Explanation of The Program:

- Plain Vanilla HelloWorld Style Program:

- **Example 1:**

1. `/* our first program!!!!*/`
2. `public class Prog1`
3. `{`
4. **`public static void main (String args[])`**
5. `{`
6. `System.out.println("Dopey");`
7. `System.out.println("Grumpy");`
8. `}`
9. `}`

- **Line 4:**

- is the heading of the method called **main**. **Every Java application must have one method called *main* which is defined *exactly* as it is on line 4.**
 - The main method is always the first method that is executed. When the program executes, main goes first.
 - This particular class is very simple and has but a single method, main().
 - Of course most classes will have many methods.

Explanation

- Plain Vanilla HelloWorld Style Program:

- **Example 1:**

- 1. `/* our first program!!!!*/`
 - 2. `public class Prog1`
 - 3. `{`
 - 4. **`public static void main (String args[])`**
 - 5. `{`
 - 6. `System.out.println("Dopey");`
 - 7. `System.out.println("Grumpy");`
 - 8. `}`
 - 9. `}`

- **Line 4:**

- The word *public* on line 4 is an access modifier for the main method.
 - If a method is declared public then that method is accessible anywhere within the program i.e. anywhere outside the class where it is defined.
 - The *private* access modifier specifies that the method is accessible only within the class.
 - If main were not public, then main could not be called from “outside” the class (i.e. by the system) and the program could not be executed.

Explanation

- Plain Vanilla HelloWorld Style Program:

- **Example 1:**

- 1. `/* our first program!!!!*/`
 - 2. `public class Prog1`
 - 3. `{`
 - 4. **`public static void main (String args[])`**
 - 5. `{`
 - 6. `System.out.println("Dopey");`
 - 7. `System.out.println("Grumpy");`
 - 8. `}`
 - 9. `}`

- **Line 4:**

- The keyword *static* signifies that a method is a **class method**.
 - A *static method* is one that can be invoked whether or not an instance of the class is created.
 - Outside of the defining class, non-static methods are invoked via objects (`r1.getArea();`) and static methods are invoked using the class name, (`Math.sqrt(64.0);`), where `Math` is a class).
 - Static methods are available whether or not any objects are created.
 - A static method belongs to the class, not to any particular instance of the class (object).
 - In a way, static methods really go against the grain of object-oriented programming, and they are very much like the functions of C, and procedures of Pascal.

Explanation

- Plain Vanilla HelloWorld

Style Program:

- **Example 1:**

```
1. /* our first program!!!!*/  
2. public class Prog1  
3. {  
4. public static void main (String  
   args[])  
5. {  
6. System.out.println("Dopey");  
7. System.out.println("Grumpy");  
8. }  
9. }
```

- **Line 4**

- The keyword *void* is used as in C++, to specify that method main returns no value.
- *String args[]* is a parameter to main.

- **Line 1 and 9 | Line 5 and 8**

- The curly braces denote the start and end of a block and are used as in C++.

Explanation

- Plain Vanilla HelloWorld

Style Program:

- **Example 1:**

1. `/* our first program!!!!*/`
2. `public class Prog1`
3. `{`
4. `public static void main (String
args[])`
5. `{`
6. **`System.out.println("Dopey");`**
7. **`System.out.println("Grumpy");`**
8. `}`
9. `}`

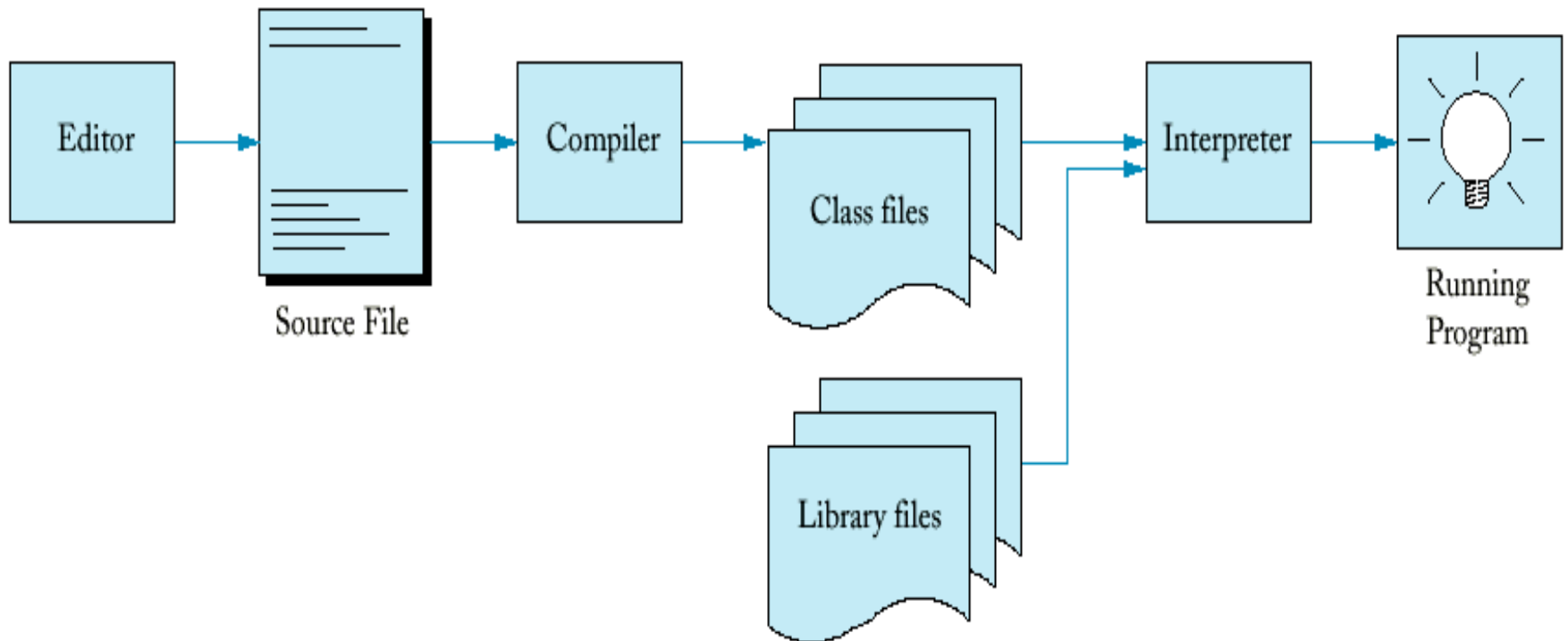
- **Line 6 and 7**

- System.out is a pre-defined system **object** associated with the standard output stream (usually the screen).
- So, in each of these statements, the System.out object invokes its println method.
- The println method accepts a string, s, as a parameter (strings are enclosed in double quotes) and sends s, followed by a carriage return, to the standard output device (the screen).
- “Dopey” and “Grumpy” are string literals.
- Notice the class Prog1 is *sending a message* to System.out. (I.e The message is “println(“Dopey”).)

Compiling and Running

- Type program into text editor
- Save
- Open command shell
- Compile into byte codes
`javac Prog1.java`
- Execute byte codes
`java Prog1`

From Source Code to Running Program



Syntax Errors vs. Logic Errors

- Syntax errors

```
System.ouch.print ("...");  
System.out.print ("Hello);
```

- Detected by the compiler

- Logic errors

```
System.out.print ("Helo");
```

- Detected (hopefully) through testing ²³

Let us try an example

- Write a program to display:
 - “Welcome to Brooklyn College”

String Concatenation

```
public class Prog3 //saved as prog3.java
{
    public static void main (String args[])
    {
        System.out.println("Dopey" + 2+3+4);
        System.out.println(2+3+4+"Dopey" );
        System.out.println("Dopey" + ( 2+3+4) )
            ;
        System.out.println(2+3+4);
        System.out.println( ( 2+3+4) );
    }
}
```

Output:

Dopey234

9Dopey

Dopey9

9

9

Here, you should notice that:

- Addition, as usual, is performed left to right.
- The argument to println is always a String.
- If the argument to println is x+y and either x or y is a string, then + **effects string concatenation**.
- For example, consider the method call println("Dopey" + 2+3+4). The integer 2,3,4 is converted to a string and the concatenated string "Dopey234" is passed to println.
- However, in the method call println(2+3+4+"Dopey"), the first/Second plus represents addition, the third effects concatenation. So, the string "9Dopey" is passed to the println method.

Selection and Iteration

*/*Reads 4 grades from the command line. Thus, the grades are stored as strings and must be converted to a numeric type before any processing can occur.*/**

```
public class Grades1
{
public static void main( String args[])
{
    char answer;
    int grade, sum = 0;
    double average;
/*Iteration Structure*/
    for (int i = 0; i < 4; i++)
        sum = sum + Integer.parseInt(args[i]) ;
        /* converts from string to int*/
        average = sum/4.0;
/*note that an explicit cast is not necessary*/
    System.out.print("Average is "+ average + " ");
}
```

```
/*Selection Structure*/
if (average >= 90)
    System.out.println('A');
else if (average >= 80)
    System.out.println('B');
else if (average >= 70)
    System.out.println('C');
else if (average > 60)
    System.out.println('D');
else
    System.out.println('F');
} //end main()
} //end class Grades1
```

Grades Program

- To run the Grades1 program:
> **java Grades1 80 90 70 60**
Notice that
 - args[0] is “80”
 - args[1] is “90”
 - args[2] is “70”
 - args[3] is “60”
- In order to calculate an average, the strings stored in args must be converted to integers. Java provides a class **Integer** with a **static** method, *parseInt(String s)*, that does exactly that -- accepts a string of digits and converts that string to an integer.
- If the string contains any nonnumeric characters, the program will crash.
- Again, notice that *parseInt* is a **static method** so it can be invoked without instantiating an Integer object.
- Finally, while loops, do-while loops and the switch statement work exactly as in C++.

Java Data Types

- Reference Types
- Primitive Data Types

Primitive Data Types

- The usual suspects:
 - byte, short, int, long
 - char
 - boolean
 - float, double
- The usual operators and rules apply mostly

Reference Types (1)

- Can't write about objects without referring to them
- Reference value: the only way to refer to an object
- Java has:
 - reference values (or just *references*)
 - reference expressions
 - reference variables
 - assignment of references

Declarations, Variables, Assignments

- C-like rules apply for the most part
- Type specifier followed by identifier list

Introduction to the Java Language

- Data Types
- Operators
- Control Flow Statements
- Arrays
- Functions -- Static Methods
- View Classes as modules

Example1: Hello.java

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

Data Types

- byte,short,int,long
- float,double
- char
- boolean
- reference
 - array
 - object

Operators

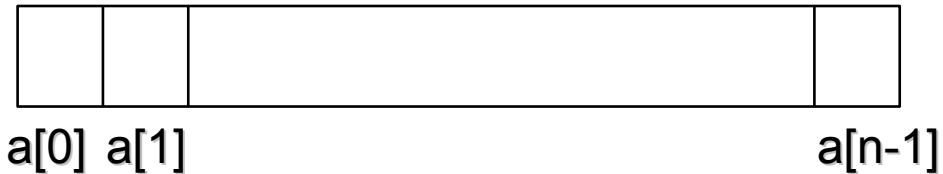
- Arithmetic operators
+, -, *, /, %, ++, --
- Relational operators
>, >=, <, <=, ==, !=
- Conditional operators
&&, ||, !
- Bitwise operators
>>, <<, >>>, &, |, ^, ~

Control Flow Statements

- decision making
 - if-else, switch-case
- loop
 - for, while, do-while
- exception
 - try-catch-finally, throw
- miscellaneous
 - break, continue, label: , return

Arrays

Type[] a = new Type[n]



- **Dynamic** **Type[] a;**
 ...
 a = new Type[expn]
- **Anonymous array**
 a = new Type[]{e1,e2,...,en};
- **a.length**

Arrays

- Superficially Similar To C:
 - [] syntax
 - start from 0
- Actually they are classless objects
- An array variable holds a reference to an array object
- Has length field— read-only

Example 2: Max.java

```
class Max {  
    public static void main(String[] args){  
        int[] a = new int[]{5,6,1,2,7,9,0,10};  
        int max;  
  
        max = a[0];  
        for (int i=1; i<a.length; i++){  
            if (a[i]>max) max = a[i];  
        }  
        System.out.println(" max="+max);  
    }  
}
```

Example 3: PrintArray

```
class PrintArray {
    public static void main(String[] args){
        int[][] a = new int[][]{{1,2,3}, {4,5,6}, {7,8,9}};
        for (int i=0; i<a.length; i++){
            for (int j=0; j< a[0].length; j++){
                System.out.print(a[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```