High-Level Programming Languages

Instructor: Dmitri A. Gusev

Spring 2007

CSC 120.02: Introduction to Computer Science

Lecture 11, March 6, 2007

Translating Programs

Assemblers translate the assembly-language instructions into machine code, or machine language. The assemblers are translating programs for low-level programming languages.

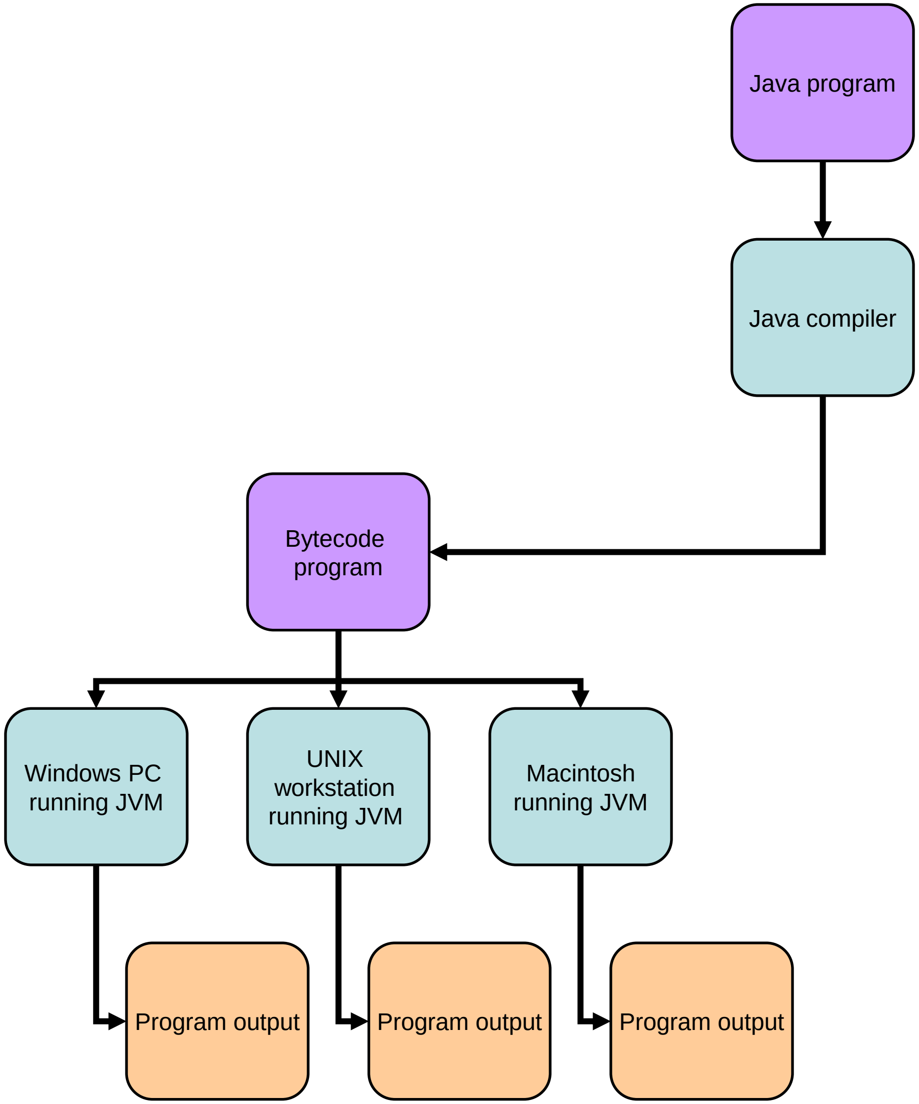
Programs that translate high-level language programs into machine code are called *compilers*. For a high-level programming language to be used on multiple types of machines, many compilers for that language are needed.

A program that translates from a low level language to a higher level one is a *decompiler*.

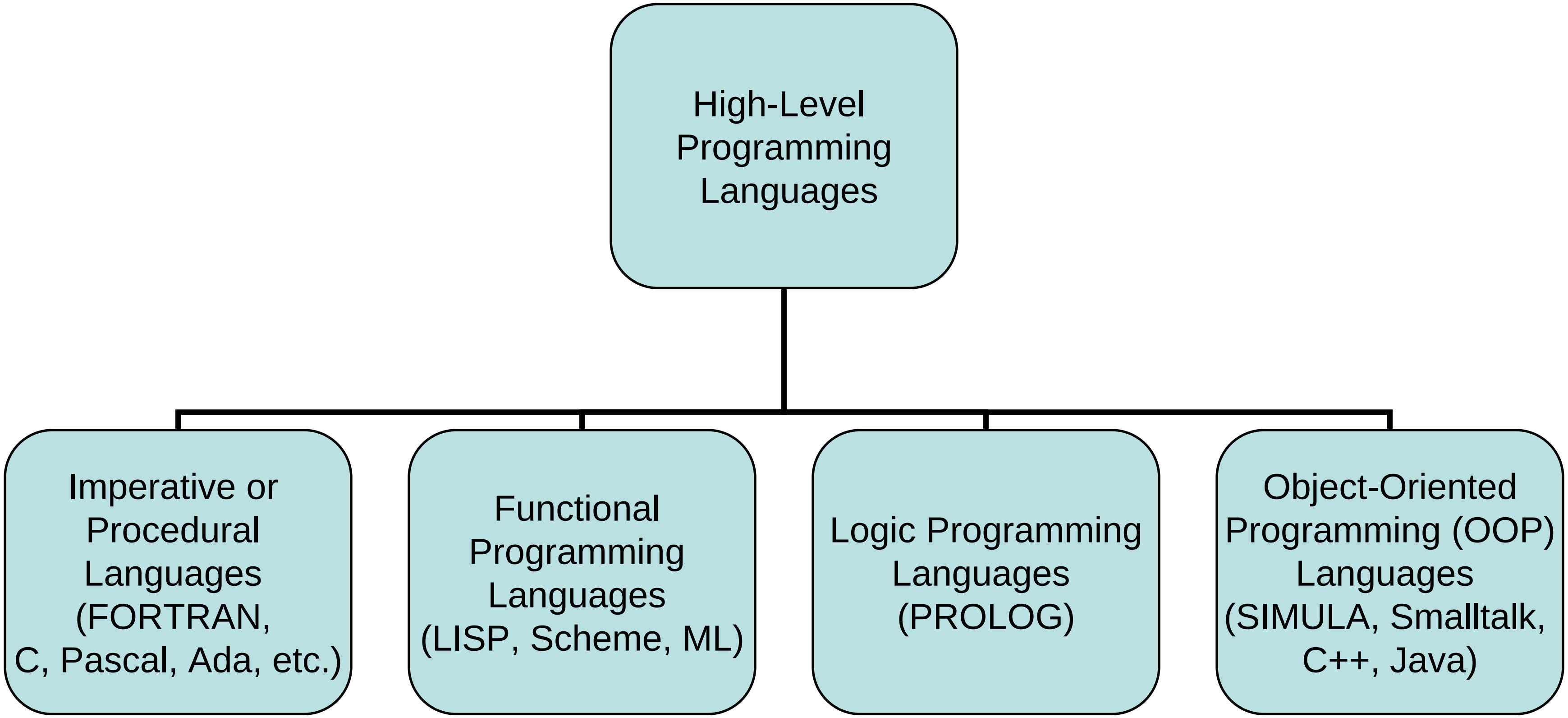
An *interpreter* is a translating program that inputs a program in a high-level language and directs the computer to immediately perform the actions specified in each statement. Interpreters can be viewed as simulators for the language in which a program is written.

Translation of Java Programs

A Java program is first *compiled* into a standard machine language called *Bytecode*. A software interpreter called the Java Virtual Machine (JVM) then takes the Bytecode program and executes it. Any machine that has a JVM can run the compiled Java program.



Classification of High-Level Programming Languages



High-Level Programming Languages

Imperative or Procedural Languages (FORTRAN, C, Pascal, Ada, etc.)

Functional Programming Languages (LISP, Scheme, ML)

Logic Programming Languages (PROLOG)

Object-Oriented Programming (OOP) Languages (SIMULA, Smalltalk, C++, Java)

Java is an object-oriented language with some imperative features. Let’s discuss these features in more detail…

Boolean Expressions

A *Boolean expression* is a sequence of identifiers, separated by compatible operators, that evaluates to *true* or *false*. A Boolean expression can be

1) a Boolean variable (its name):

```
boolean headlightsOn; // headlightsOn is declared here.
/* headlightsOn is a Boolean variable…
   This was a multi-line comment. */
if (headlightsOn) // headlightsOn is a Boolean expression HERE!
   System.out.println(“Please turn off the lights! Your battery.”);
```

Boolean Expressions (cont'd)

A Boolean expression can also be

2) an arithmetic expression followed by a relational operator followed by an arithmetic expression. *Relational operators*, a.k.a. *conditional operators*, are:

Operator	Name
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Boolean Expressions (cont’d)

A Boolean expression can also be

3) A Boolean expression followed by a Boolean operator followed by a Boolean expression. The Boolean operators are:

Operator	Name
!	NOT
&&	Conditional-AND
||	Conditional-OR

Data Types

In Java, there are 8 *primitive data types*:

1) byte — 8-bit signed two's complement integer
2) short —16-bit signed two's complement integer
3) int — 32-bit signed two's complement integer
4) long — 64-bit signed two's complement integer
5) float — 32-bit floating point
6) double — 64-bit floating point
7) boolean — only two possible values: true and false
8) char — a single 16-bit Unicode character

In addition to that, special support for character strings is provided:

```
String s = “this is the true name of Thoth”;
// Once created, the values of String objects cannot be changed!
```

Strong typing means that each variable is assigned a type, and only values of that type can be stored in the variable.

A *data type* is a description of the set of values and the basic set of operations that can be applied to values of the type.

Declarations

A *declaration* is a statement that associates an identifier with a variable, an action, or some other entity within the language.

```
// field declaration in Java

private int numberOfSecretChambers; // A field declaration with a MODIFIER!
```

In Java, the following kinds of *variables* are defined:

1) Instance variables (non-static fields)
2) Class variables (static fields): They are declared with the static *modifier*
3) Local variables: They are only visible to the methods in which they are declared
4) Parameters

Access Levels in Java

The following table shows the access to fields and methods permitted by each *modifier*.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Assignment Statement

An *assignment statement* is a statement that stores the value of an expression in a variable.

```
public class Circle
{
    private double radius;

    public Circle() // Default constructor
    {
        radius = 1.0; // This is an assignment statement
    }

    public Circle(double r) // Construct a circle with a specified radius
    {
        radius = r; // This is yet another assignment statement
    }

    public double findArea()
    {
        return radius*radius*3. 14159265358979;
    }
}
```

The if Statement

```
if (height<0.0)
    System.out.println(“This is an inverted pyramid, Dan!”);
else if (height>0.0) // a “nested” if statement
    { // begin block
        System.out.println(“This is a normal pyramid.”);
        System.out.println(“Yup. A regular one.”)
    } // end block
else
    System.out.println(“No pyramid found.”);
```

The switch Statement

```
class SwitchDemo
{
    public static void main(String[] args)
    {
        int pyramidID = 3;

        switch (pyramidID)
        {
          case 1: System.out.println("Pi-ramid"); break;
          case 2: System.out.println("Py-thagorean triangle"); break;
          case 3: System.out.println("Golden ratio"); break;
          case 4: System.out.println("Lady with a False Beard"); break;
          default: System.out.println("No such pyramid in Giza"); break;
        }
    }
}
```

Looping Statements

```
/*--- 1 ---*/
while (expression) // The while loop begins here
{
    statement(s)
} // end of the while loop

/*--- 2 ---*/
do // The do-while loop begins here
{
    statement(s)
}
while (expression); // end of the do-while loop
```

Looping Statements (cont'd)

```
class ForDemo
{
    public static void main(String[] args)
    {       /*--- 3 ---*/
            for(int i=1; i<5; i++) // i is a local variable! The for loop begins here
            {
              System.out.println("Count is: " + i);
            } // end of the for loop
    }
 }
```

The *initialization* expression initializes the loop; it's executed once, as the loop begins.

When the *termination* expression evaluates to false, the loop terminates.

The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to *increment* or *decrement* a value.

Arrays

```
int[] anArray; // declares an array of integers
anArray = new int[10]; // allocates memory for 10 integers
anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element

anArray[2] = 300; // etc.
```

Recursion

```
// Recursive method for computing factorial of n
static long factorial(int n)
{
    if (n == 0) // Stopping condition
        return 1;
    else
        return n*factorial(n-1); // Call factorial recursively
}
```