Program Elements -- Introduction

- We can now examine the core elements of programming
- Chapter 3 focuses on:
  - data types
  - variable declaration and use
  - operators and expressions
  - decisions and loops
  - input and output

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Primitive Data Types

- A *data type* is defined by a set of values and the operators you can perform on them
- Each value stored in memory is associated with a particular data type
- The Java language has several predefined types, called *primitive data types*
- The following reserved words represent eight different primitive types:
  - `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Integers

- There are four separate integer primitive data types
- They differ by the amount of memory used to store them

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Floating Point

- There are two floating point types:

Type	Storage	Approximate Min Value	Approximate Max Value
float	32 bits	-3.4×10^{38}	3.4×10^{38}
double	64 bits	-1.7×10^{308}	1.7×10^{308}

- The `float` type stores 7 significant digits
- The `double` type stores 15 significant digits

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Characters

- A `char` value stores a single character from the *Unicode character set*
- A *character set* is an ordered list of characters
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Characters

- The ASCII character set is still the basis for many other programming languages
- ASCII is a subset of Unicode, including:

uppercase letters	A, B, C, …
lowercase letters	a, b, c, …
punctuation	period, semi-colon, …
digits	0, 1, 2, …
special symbols	&, |, \, …
control characters	carriage return, tab, ...

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Boolean

- A `boolean` value represents a true or false condition
- They can also be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a boolean type

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Wrappers

- For each primitive data type there is a corresponding *wrapper class*. For example:

Primitive Type	Wrapper Class
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

- Wrapper classes are useful in situations where you need an object instead of a primitive type

- They also contain some useful methods

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Variables

- A *variable* is an identifier that represents a location in memory that holds a particular type of data
- Variables must be declared before they can be used
- The syntax of a variable declaration is:

```
data-type variable-name;
```

- For example:

```
int total;
```


Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Variables

- Multiple variables can be declared on the same line:

```
int total, count, sum;
```

- Variables can be *initialized* (given an initial value) in the declaration:

```
int total = 0, count = 20;

float unit_price = 57.25;
```

- See `Piano_Keys.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Assignment Statements

- An assignment statement takes the following form:

```
variable-name = expression;
```

- The expression is evaluated and the result is stored in the variable, overwriting the value currently stored in the variable
- See `United_States.java`
- The expression can be a single value or a more complicated calculation

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Constants

- A constant is similar to a variable except that they keep the same value throughout their existence
- They are specified using the reserved word `final` in the declaration
- For example:

```
final double PI = 3.14159;

final int STUDENTS = 25;
```


Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Constants

- When appropriate, constants are better than variables because:
  - they prevent inadvertent errors because their value cannot change
- They are better than literal values because:
  - they make code more readable by giving meaning to a value
  - they facilitate change because the value is only specified in one place

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Input and Output

- Java I/O is based on *input streams* and *output streams*
- There are three predefined standard streams:

Stream	Purpose	Default Device
`System.in`	reading input	keyboard
`System.out`	writing output	monitor
`System.err`	writing errors	monitor

- The `print` and `println` methods write to standard output

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Input and Output

- The Java API allows you to create many kinds of streams to perform various kinds of I/O
- To read character strings, we will convert the `System.in` stream to another kind of stream using:

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

- This declaration creates a new stream called `stdin`
- We will discuss object creation in more detail later

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Escape Sequences

- See `Echo.java`
- An *escape sequence* is a special sequence of characters preceded by a backslash (\)
- They indicate some special purpose, such as:

Escape Sequence	Meaning
`\t`	tab
`\n`	new line
`\"`	double quote
`\'`	single quote
`\\`	backslash

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Buffers

- As you type, the characters are stored in an *input buffer*
- When you press enter, the program begins processing the data
- Similarly, output information is temporarily stored in an *output buffer*
- The output buffer can be explicitly *flushed* (sent to the screen) using the `flush` method
- See `Python.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Numeric Input

- Converting a string that holds an integer into the integer value can be done with a method in the `Integer` wrapper class:

```
value = Integer.parseInt (my_string);
```

- A value can be read and converted in one line:

```
num = Integer.parseInt (stdin.readLine());
```

- See `Addition.java` and `Addition2.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Expressions

- An *expression* is a combination of operators and operands
- The arithmetic operators include addition (+), subtraction (–), multiplication (*), and division (/)
- Operands can be literal values, variables, or other sources of data
- The programmer determines what is done with the result of an expression (stored, printed, etc.)

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Division

- If the operands of the `/` operator are both integers, the result is an integer (the fractional part is truncated)
- If one or more operands to the `/` operator are floating point values, the result is a floating point value
- The remainder operator (`%`) returns the integer remainder after dividing the first operand by the second
- The operands to the `%` operator must be integers
- See `Division.java`
- The remainder result takes the sign of the numerator

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Division

Expression	Result
`17 / 5`	`3`
`17.0 / 5`	`3.4`
`17 / 5.0`	`3.4`
`9 / 12`	`0`
`9.0 / 12.0`	`0.75`
`6 % 2`	`0`
`14 % 5`	`4`
`-14 % 5`	`-4`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Operator Precedence

- The order in which operands are evaluated in an expression is determined by a well-defined *precedence hierarchy*
- Operators at the same level of precedence are evaluated according to their *associativity* (right to left or left to right)
- Parentheses can be used to force precedence
- Appendix D contains a complete operator precedence chart for all Java operators

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Operator Precedence

- Multiplication, division, and remainder have a higher precedence than addition and subtraction
- Both groups associate left to right

Expression: `5 + 12 / 5 - 10 % 3`

Order of evaluation: 3 1 4 2

Result: 6

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Operator Precedence

Expression	Result
`2 + 3 * 4 / 2`	8
`3 * 13 + 2`	41
`(3 * 13) + 2`	41
`3 * (13 + 2)`	45
`4 * (11 - 6) * (-8 + 10)`	40
`(5 * (4 - 1)) / 2`	7

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The if Statement

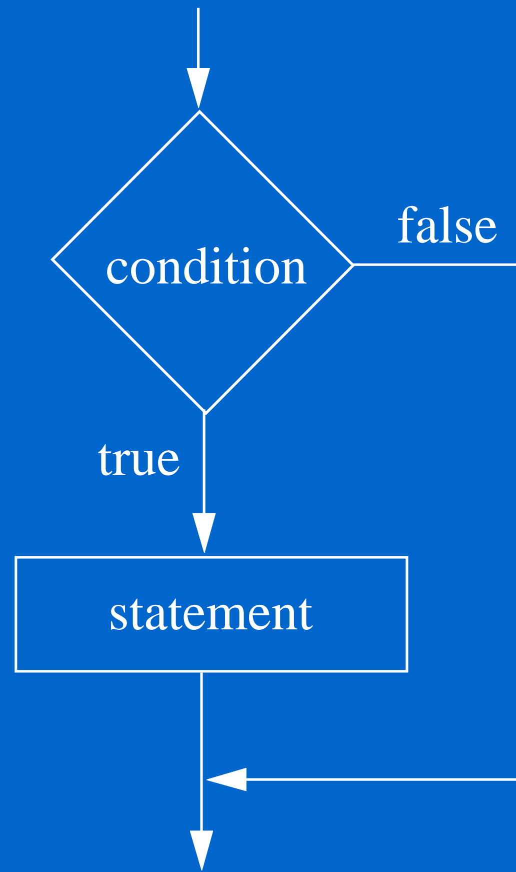
- The Java *if statement* has the following syntax:

```
if (condition)
    statement;
```

- If the boolean condition is true, the statement is executed; if it is false, the statement is skipped
- This provides basic decision making capabilities

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The if Statement



Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Boolean Expressions

- The condition of an `if` statement must evaluate to a true or false result
- Java has several equality and relational operators:

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

- See `Temperature.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Block Statements

- Several statements can be grouped together into a *block statement*
- Blocks are delimited by braces
- A block statement can be used wherever a statement is called for in the Java syntax
- See `Temperature2.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The if-else Statement

- An `else` clause can be added to an `if` statement to make it an *if-else statement*:

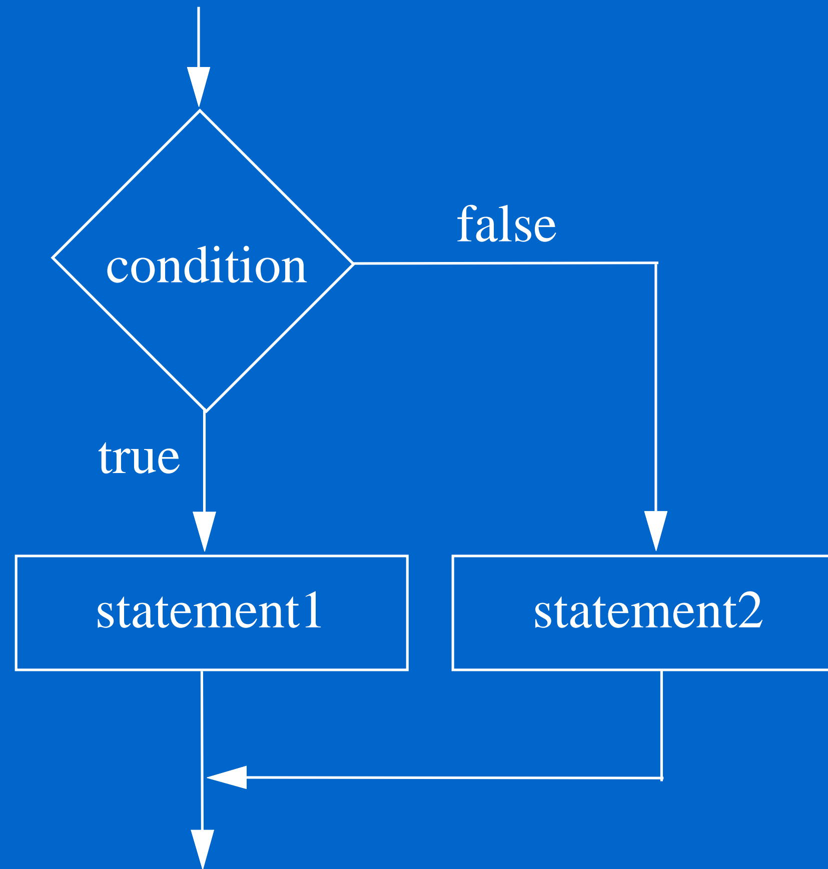
```
if (condition)
    statement1;
else
    statement2;
```

- If the condition is true, statement1 is executed; if the condition is false, statement2 is executed

- See `Temperature3.java` and `Right_Triangle.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The if-else Statement



Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Nested if Statements

- The body of an `if` statement or `else` clause can be another `if` statement
- These are called *nested if statements*
- See `Football_Choice.java`
- Note: an `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The while Statement

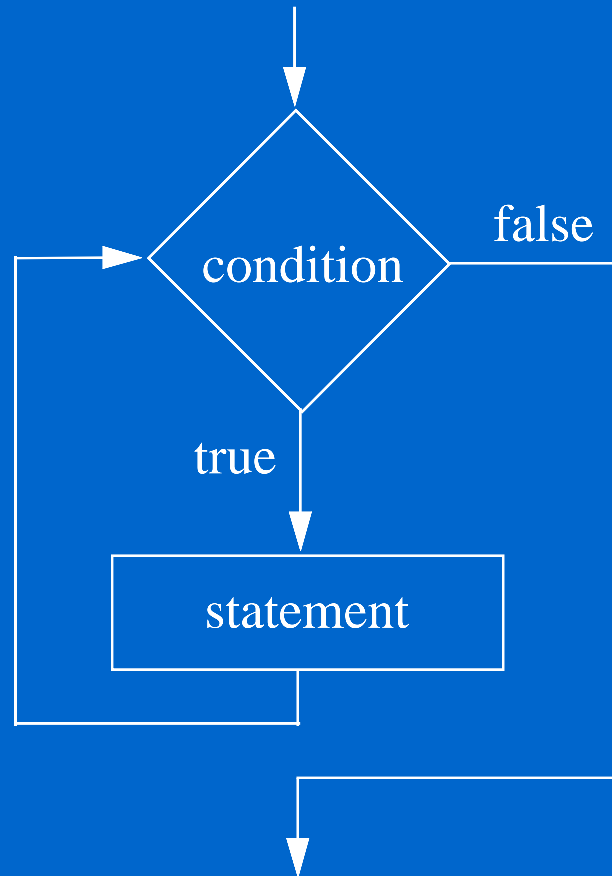
- A *while statement* has the following syntax:

```
while (condition)
    statement;
```

- If the condition is true, the statement is executed; then the condition is evaluated again
- The statement is executed over and over until the condition becomes false

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The while Statement



Copyright 1997 by John Lewis and William Loftus. All rights reserved.

The while Statement

- If the condition of a `while` statement is false initially, the statement is never executed
- Therefore, we say that a `while` statement executes zero or more times
- See `Counter.java`, `Factors.java`, and `Powers_of_Two.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Infinite Loops

- The body of a `while` loop must eventually make the condition false
- If not, it is an *infinite loop*, which will execute until the user interrupts the program
- This is a common type of logical error -- always double check that your loops will terminate normally
- See `Forever.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Program Development

- The creation of software involves four basic activities:
  - establishing the requirements
  - creating a design
  - implementing the code
  - testing the implementation

- The development process is much more involved that this, but these basic steps are a good starting point

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Requirements

- Requirements specify the tasks a program must accomplish (what to do, not how to do it)
- They often address the user interface
- An initial set of requirements are often provided, but usually must be critiqued, modified, and expanded
- It is often difficult to establish detailed, unambiguous, complete requirements
- Careful attention to the requirements can save significant time and money in the overall project

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Design

- A program follows an *algorithm*, which is a step-by-step process for solving a problem
- The design specifies the algorithms and data needed
- In object-oriented development, it establishes the classes, objects, and methods that are required
- The details of a method may be expressed in *pseudocode*, which is code-like, but does not necessarily follow any specific syntax

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Implementation

- Implementation is the process of translating a design into source code
- Most novice programmers think that writing code is the heart of software development, but it actually should be the least creative
- Almost all important decisions are made during requirements analysis and design
- Implementation should focus on coding details, including style guidelines and documentation

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Testing

- A program should be executed multiple times with various input in an attempt to find errors
- *Debugging* is the process of discovering the cause of a problem and fixing it
- Programmers often erroneously think that there is "only one more bug" to fix
- Tests should focus on design details as well as overall requirements

Copyright 1997 by John Lewis and William Loftus. All rights reserved.

Program Development

- See `Average.java`
- Follow the process of requirements analysis, design, implementation, and testing
- There are always multiple ways to design and implement a program
- Any design has advantages and disadvantages; there are always trade-offs
- See `Average2.java`

Copyright 1997 by John Lewis and William Loftus. All rights reserved.