Chapter 1 - Java Fundamentals

Chapter 1 - Java Fundamentals

Chapter 1 - Java Fundamentals

Chapter Goals

- ✓ To become familiar with your computing environment and your compiler
- ✓ To compile and run your first Java program
- ✓ To recognize syntax and logic errors
- ✓ To write pseudocode for simple algorithms
- ✓ To learn about Data and Expressions
- ✓ To learn about Predefined classes
- ✓ To learn about Conditionals and Loops
- ✓ To learn about Arrays

What Is Programming?

✓Computers are programmed to perform tasks

✓Different tasks = different programs

✓Program

  ▪Sequence of basic operations executed in succession

  ▪Contains instruction sequences for all tasks it can execute

✓Sophisticated programs require teams of highly skilled programmers and other professionals

The Anatomy of a Computer

✓Central processing unit
- Chip
- Transistors

✓Storage
- Primary storage: Random-access memory (RAM)
- Secondary storage: e.g. hard disk
- Removable storage devices: e.g.: floppy disks, tapes, CDs

✓Peripherals

✓Executes very simple instructions

✓Executes instructions very rapidly
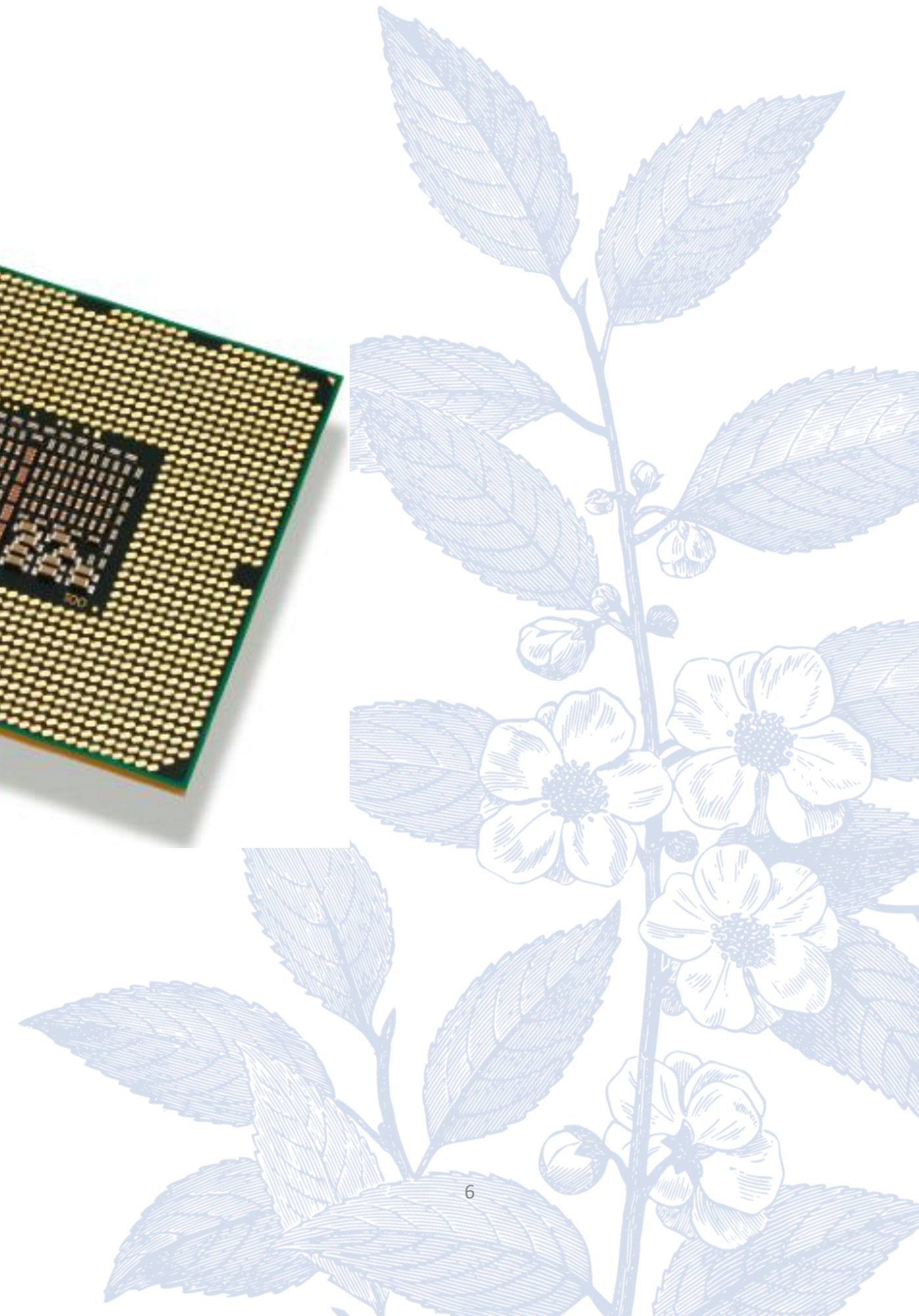
✓General purpose device

Central Processing Unit
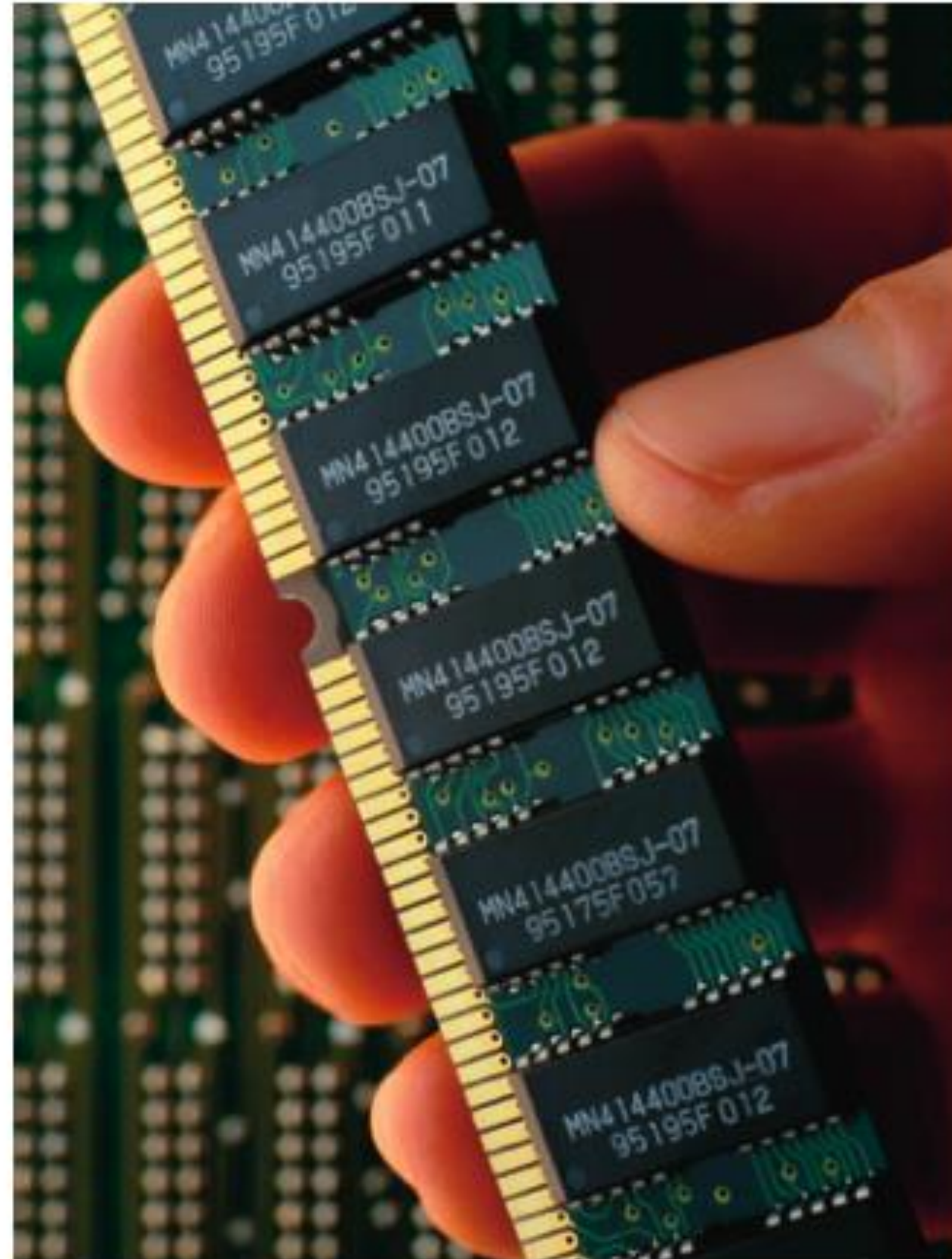

Figure 1
Central Processing Unit

A Memory Module with Memory Chips

Figure 2
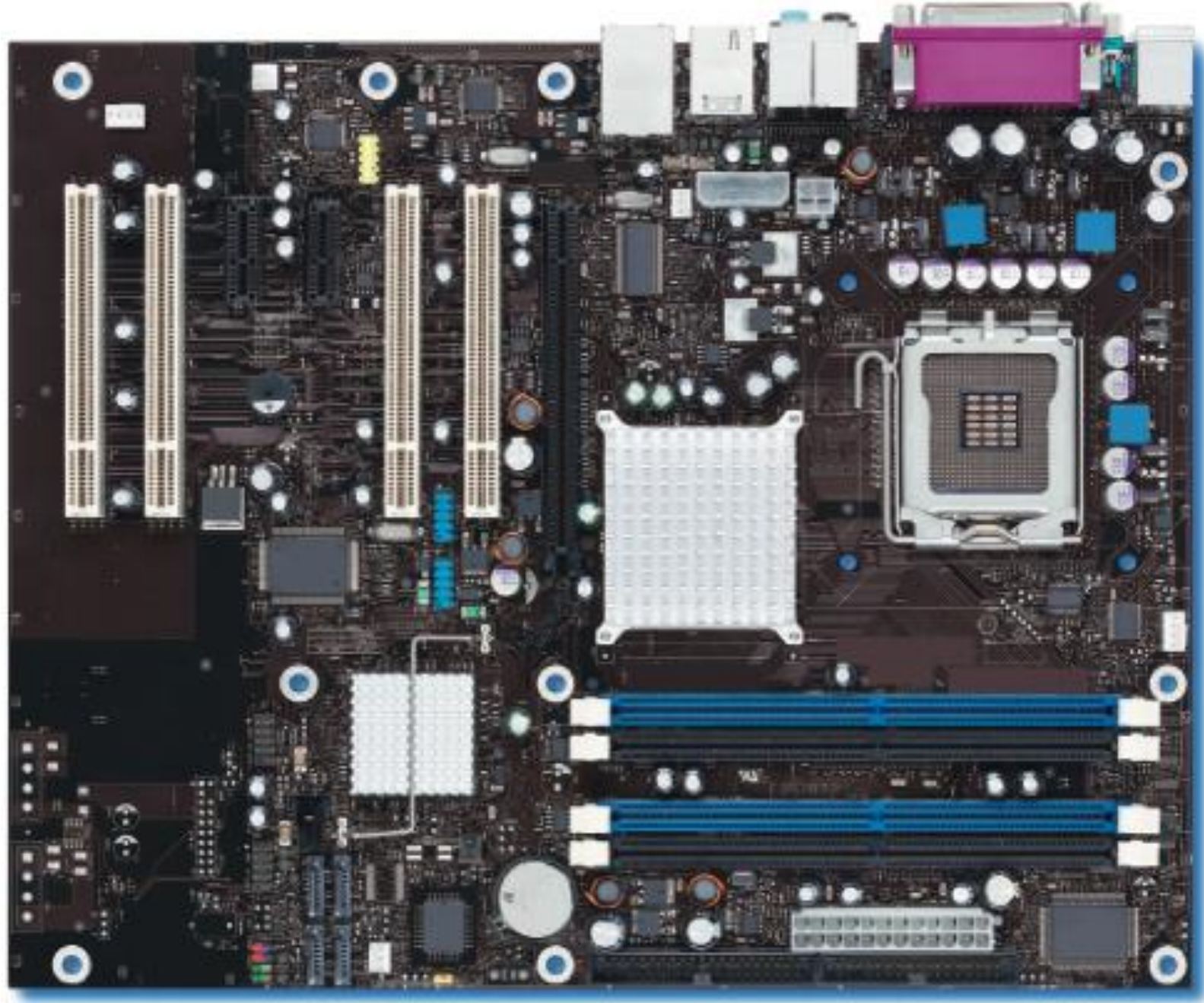A Memory Module with Memory Chips

A Hard Disk
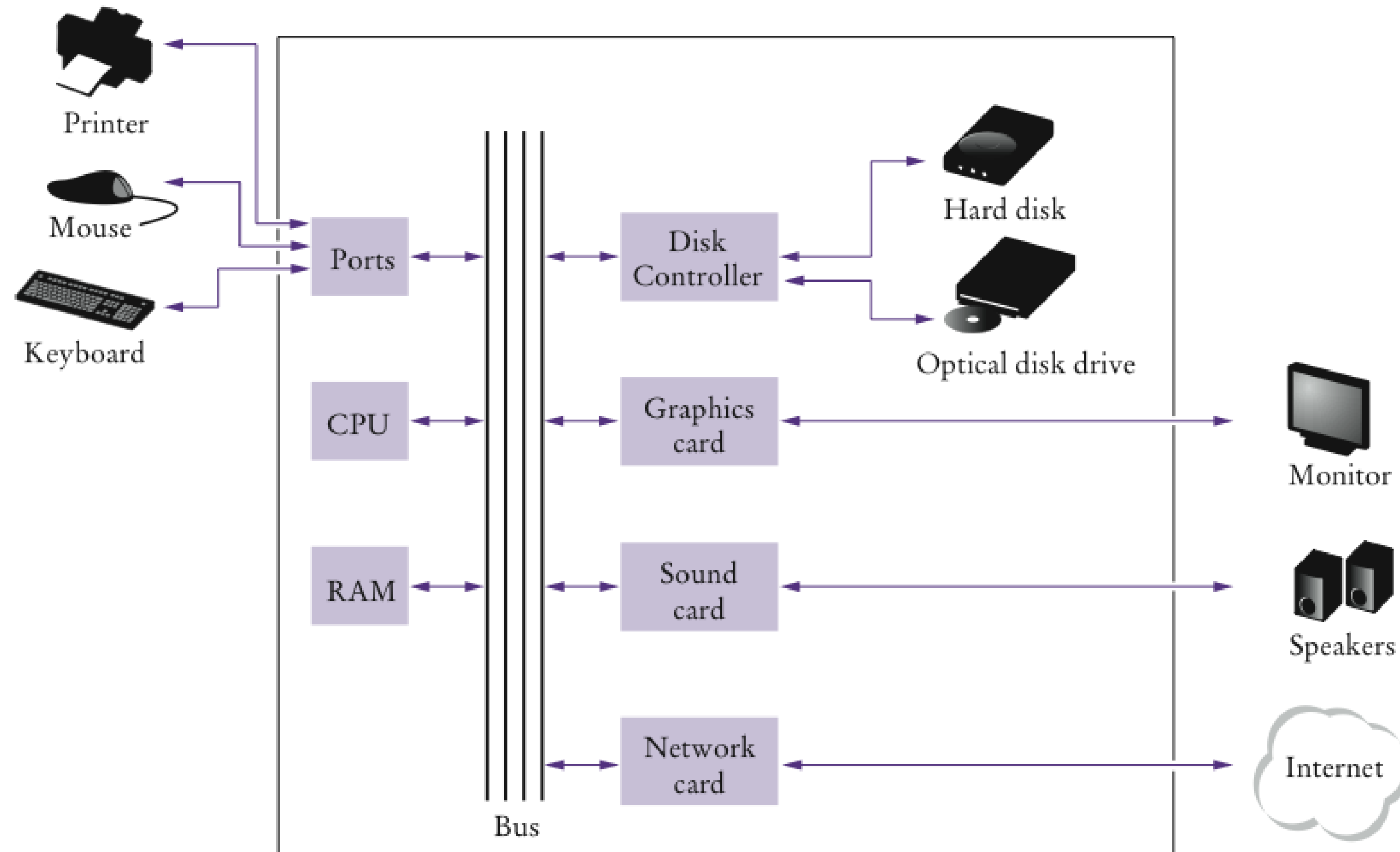

Figure 3 A Hard Disk

A Motherboard

Figure 4 A Motherboard

Schematic Diagram of a Computer

Figure 5 Schematic Diagram of a Computer

The ENIAC
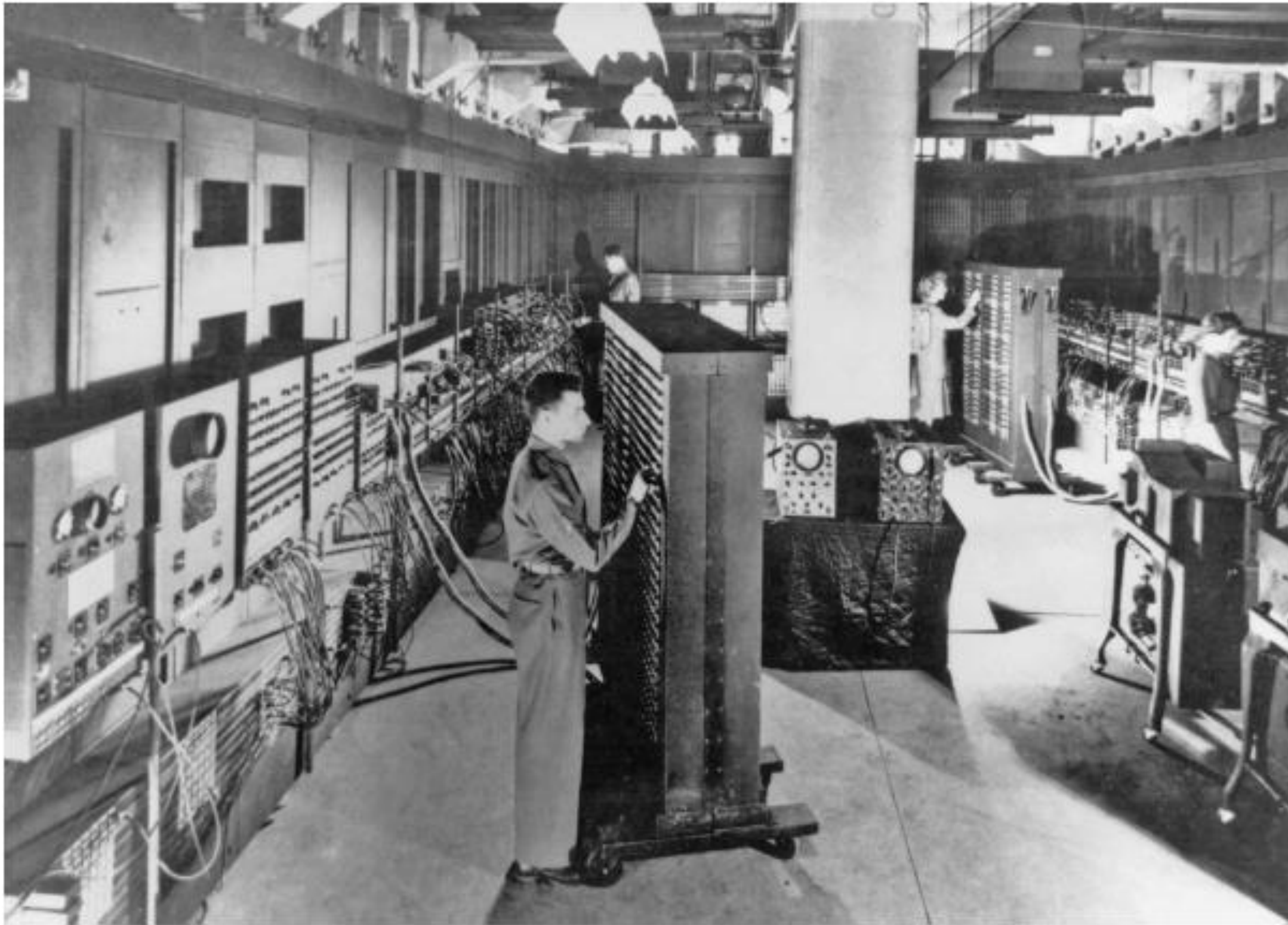
The ENIAC

Machine Code

✓Generally, machine code depends on the CPU type

✓However, the instruction set of the Java virtual machine (JVM) can be executed on many types of CPU

✓Java Virtual Machine (JVM) – a typical sequence of machine instructions is:

- 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 that is stored in memory location 240.

The Java Programming Language

✓Simple

✓Safe

✓Platform-independent ("write once, run anywhere")

✓Rich library (packages)

✓Designed for the internet
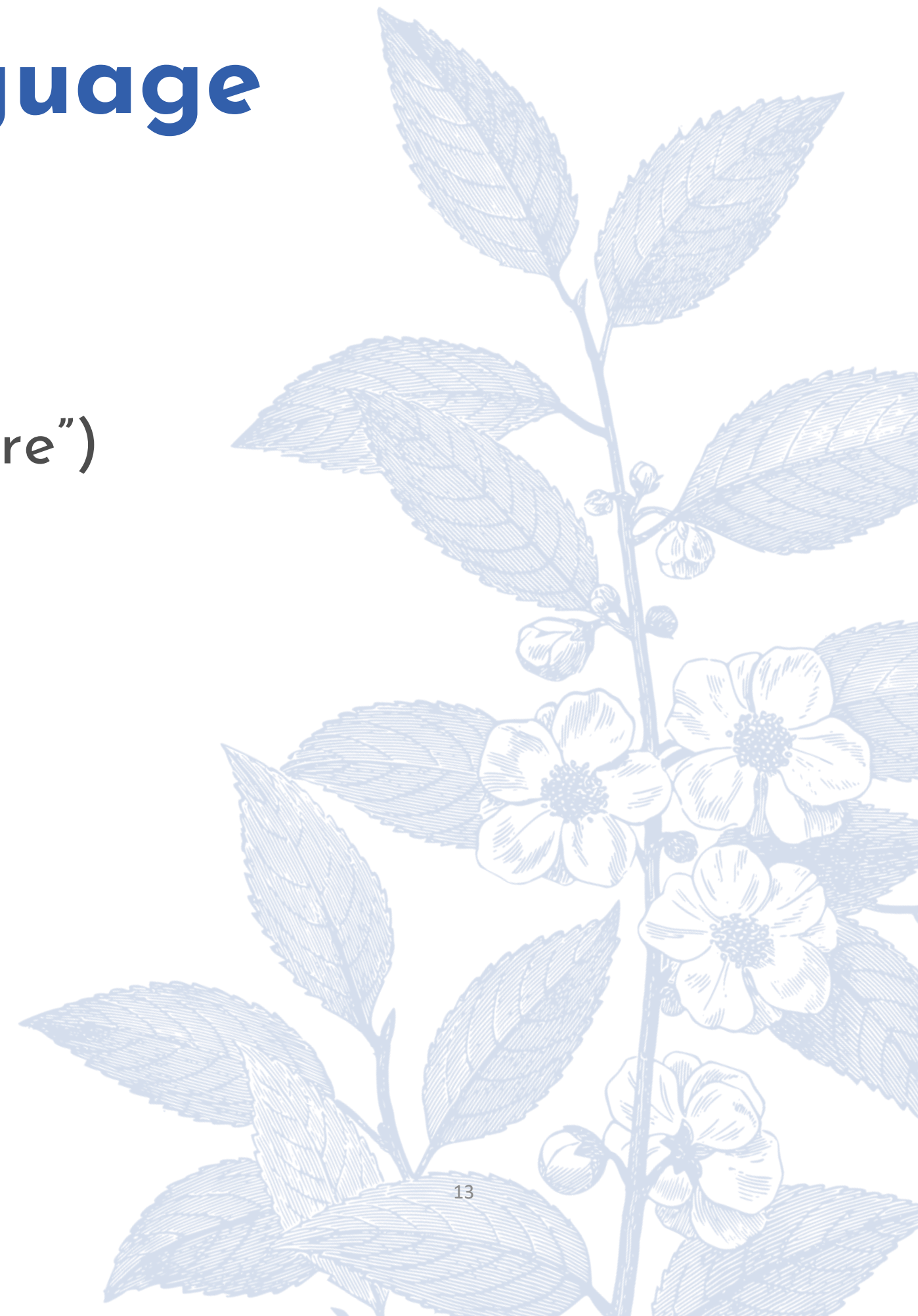
Java Versions

Version	Year	Important New Features	Version	Year	Important New Features
1.1	1997	Inner classes	12	2019	Switch Expressions, Compact Number Formatting
1.2	1998	Swing, Collections framework	13	2019	Text Blocks, New Methods in String Class for Text Blocks, Switch Expressions, Enhancements
1.3	2000	Performance enhancements	14	2020	Pattern Matching for instanceof, Helpful NullPointerExceptions, Records
1.4	2002	Assertions, XML support	15	2020	Sealed Classes, Hidden Classes, Deprecate RMI Activation for Removal
5	2004	Generic classes, enhanced for loop, auto-boxing, enumerations, annotations	16	2021	Vector API
6	2006	Library improvements	17	2021	Dismiss the Applet API for Removal, Activation of the Removal RMI, Generate sealed Classes, Removal of the Experimental AOT and JIT Compiler, Remove the Security Manager.
7	2011	Small language changes and library improvements	18	2022	UTF-8 by Default, Simple Web Server, Pattern Matching Improvements
8	2014	Function expressions, streams, new date/time library	19	2022	Virtual Threads, Record Patterns
9	2017	Modules	20	2023	structured concurrency, scoped values
10, 11	2018	Versions with incremental improvements are released every six months	21	2023	Sequenced Collections, String Templates

Parts of a Java Program

✓A Java source code file contains one or more Java classes.

✓If more than one class is in a source code file, only one of them may be public.

✓The public class and the filename of the source code file must match.

- Ex: A class named Simple must be in a file named Simple.java

✓Each Java class can be separated into parts.

✓Every Java application contains a class with a main method

- When the application starts, the instructions in the main method are executed

Compiling and Running a Java Program

✓The Java compiler translates source code into class files that contain instructions for the Java virtual machine

✓A class file has extension .class

✓The compiler does not produce a class file if it has found errors in your program

✓The Java virtual machine loads instructions from the program's class file, starts the program, and loads the necessary library files as they are required

From Source Code to Running Program
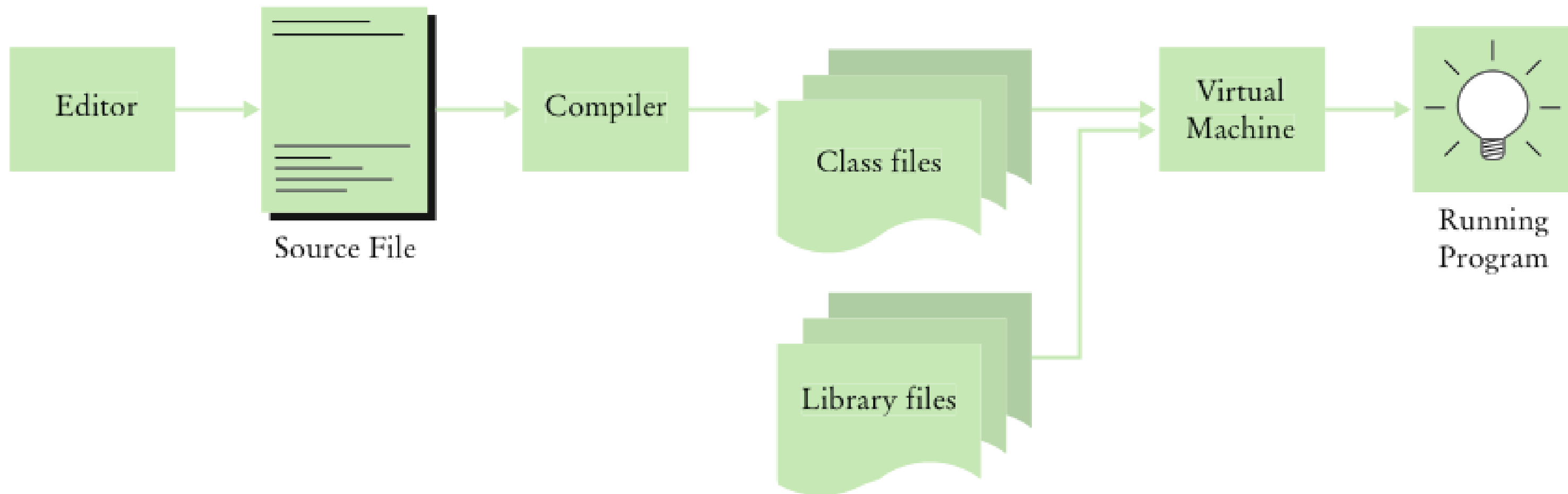
Figure 10 From Source Code to Running Program

Errors

✓Compile-time error: A violation of the programming language rules that is detected by the compiler

- Ex:

  System.ou.println("Hello, World!);

  –> Syntax error

✓Run-time error: Causes the program to take an action that the programmer did not intend
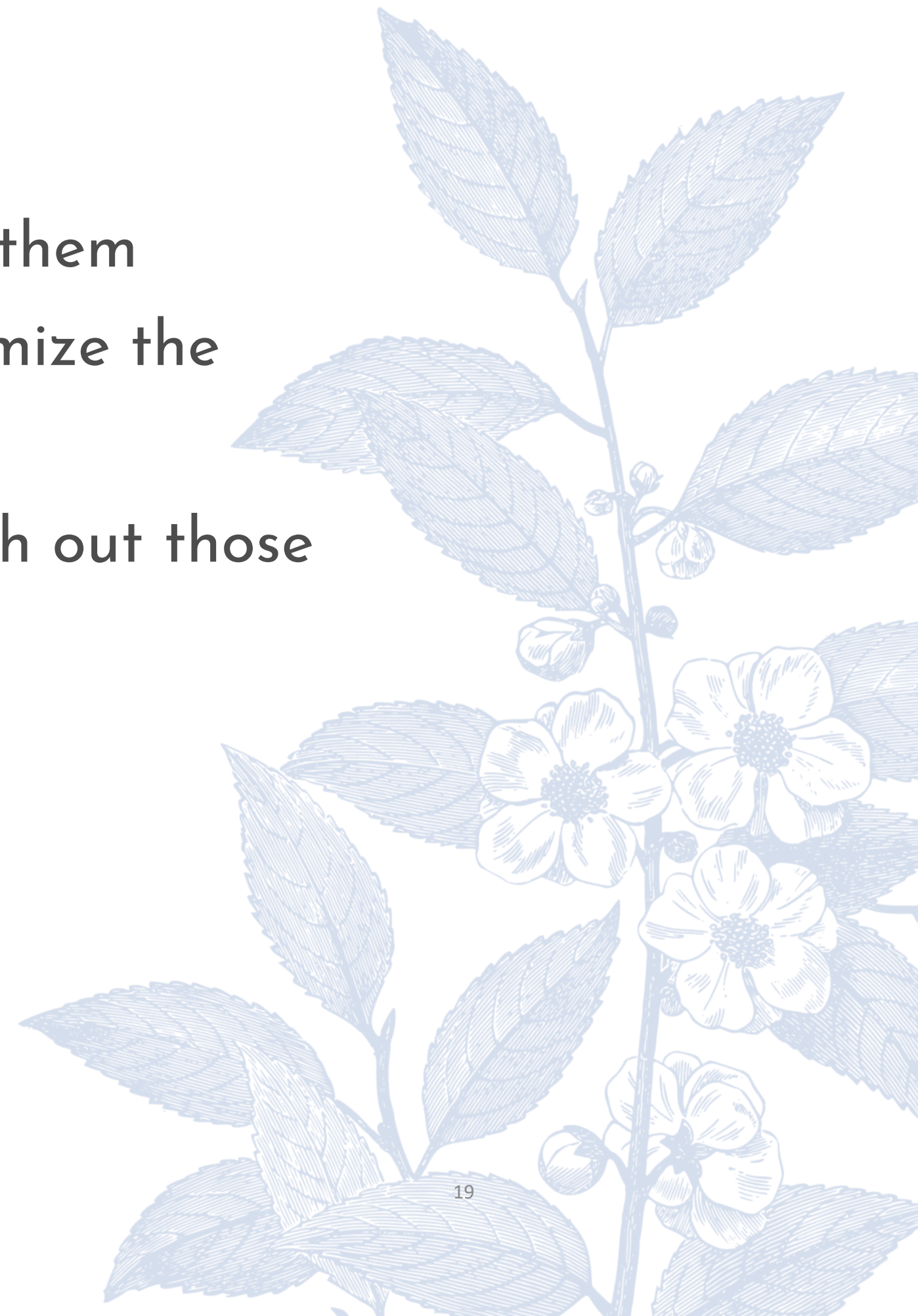
- Ex:

  System.out.println("Hello, Word!");

  System.out.println(1/0);

  –> Logic error

Error Management Strategy

✓Learn about common errors and how to avoid them

✓Use defensive programming strategies to minimize the likelihood and impact of errors

✓Apply testing and debugging strategies to flush out those errors that remain

Algorithms

✓Algorithm: A sequence of steps that is:
- unambiguous
- executable
- terminating

✓Algorithm for deciding which car to buy, based on total costs:

For each car, compute the total cost as follows:

annual fuel consumed = annual miles driven / fuel efficiency

annual fuel cost = price per gallon x annual fuel consumed

operating cost = 10 x annual fuel cost

total cost = purchase price + operating cost

If total cost1 < total cost2

Choose car1

Else

Choose car2

Pseudocode

Pseudocode: An informal description of an algorithm:

✓Describe how a value is set or changed:

total cost = purchase price + operating cost

✓Describe decisions and repetitions:

For each car

operating cost = 10 x annual fuel cost

total cost = purchase price + operating cost

Use indentation to indicate which statements should be selected or repeated

✓Indicate results:

Choose car1

Program Development Process

Understand the problem

↓

Develop and describe an algorithm

↓

Test the algorithm with different inputs

↓

Translate the algorithm into Java

↓

Compile and test your program
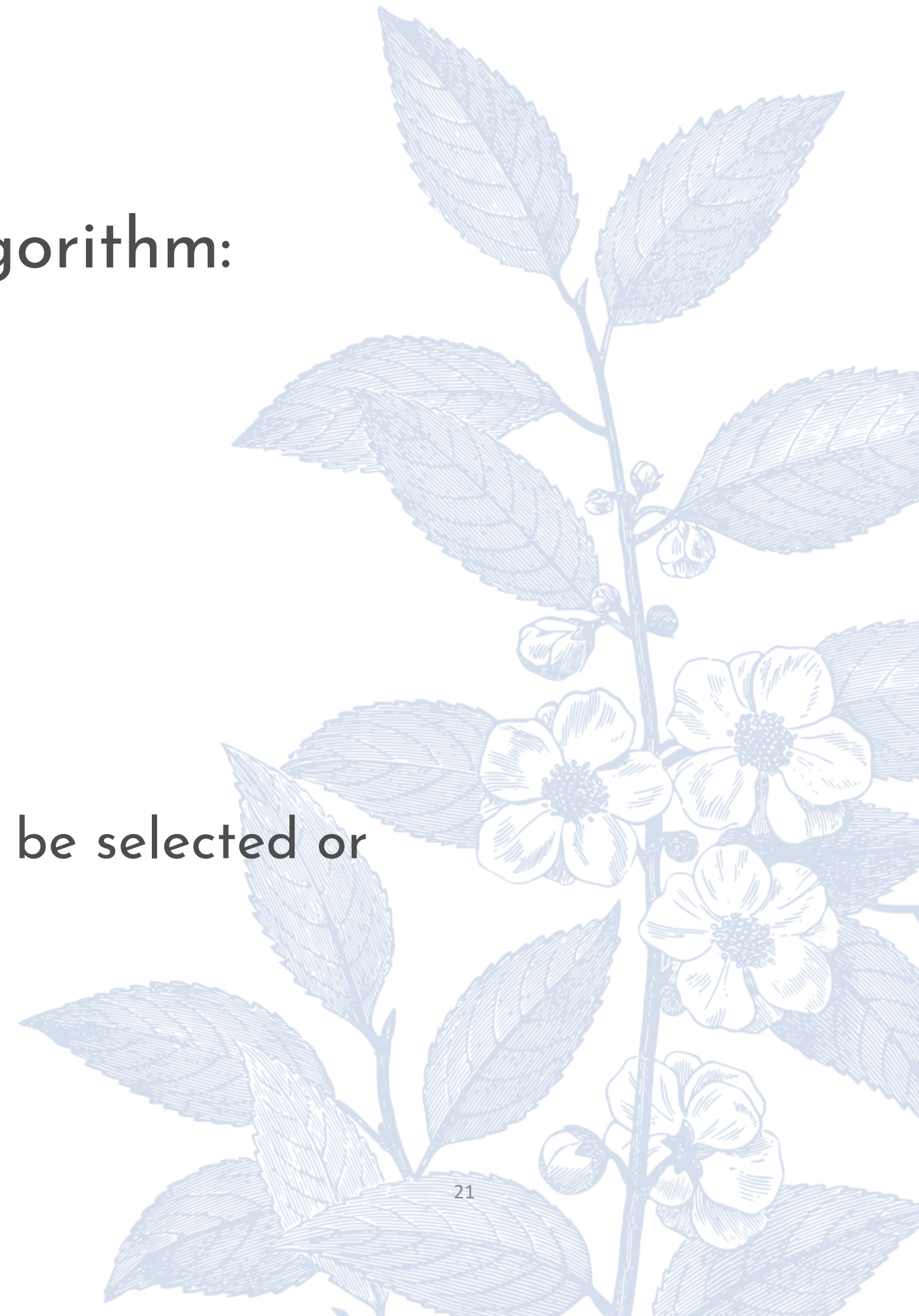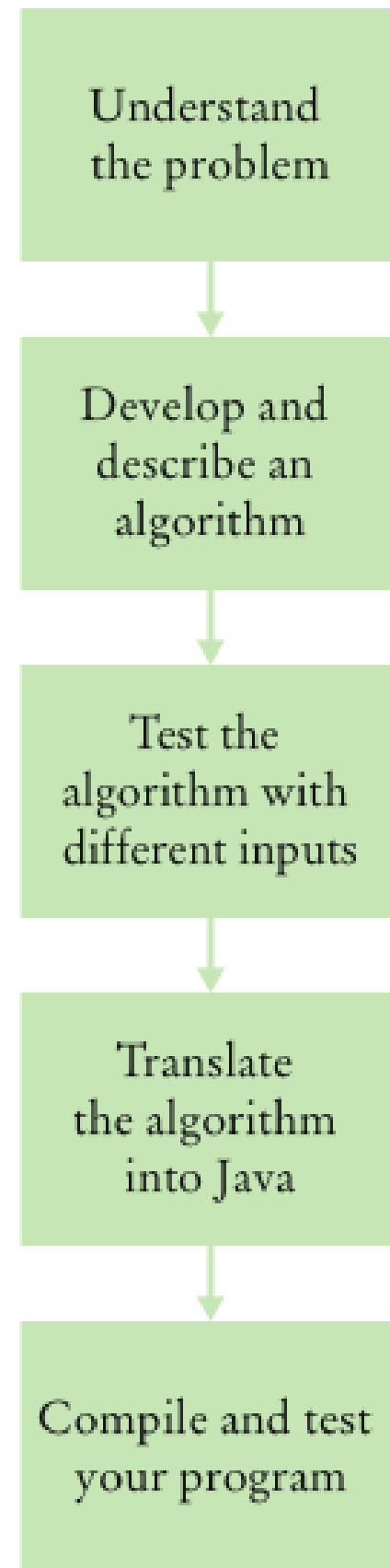
Figure 12
The Program Development Process

22

Self Check 1.18

Investment Problem: You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

Algorithm:

    Start with a year value of 0 and a balance of $10,000.

    Repeat the following steps while the balance is less than $20,000.

        Add 1 to the year value.

        Multiply the balance value by 1.05 (a 5 percent increase).

Suppose the interest rate was 20 percent. How long would it take for the investment to double?

Answer: 4 years:

    0   10,000

    1   12,000

    2   14,400

    3   17,280

    4   20,736

Self Check 1.19

Suppose your cell phone carrier charges you $29.95 for up to 300 minutes of calls, and $0.45 for each additional minute, plus 12.5 percent taxes and fees. Give an algorithm to compute the monthly charge for a given number of minutes.

Answer:

Is the number of minutes at most 300?

✓If so, the answer is $29.95 × 1.125 = $33.70.

✓If not,

- Compute the difference: (number of minutes) – 300.
- Multiply that difference by 0.45.
- Add $29.95.
- Multiply the total by 1.125. That is the answer.

Chapter 1 - Java Fundamentals

Chapter Goals

✓To become familiar with your computing environment and your compiler

✓To compile and run your first Java program

✓To recognize syntax and logic errors

✓To write pseudocode for simple algorithms

✓To learn about Data and Expressions

✓To learn about Predefined classes

✓To learn about Conditionals and Loops

✓To learn about Arrays
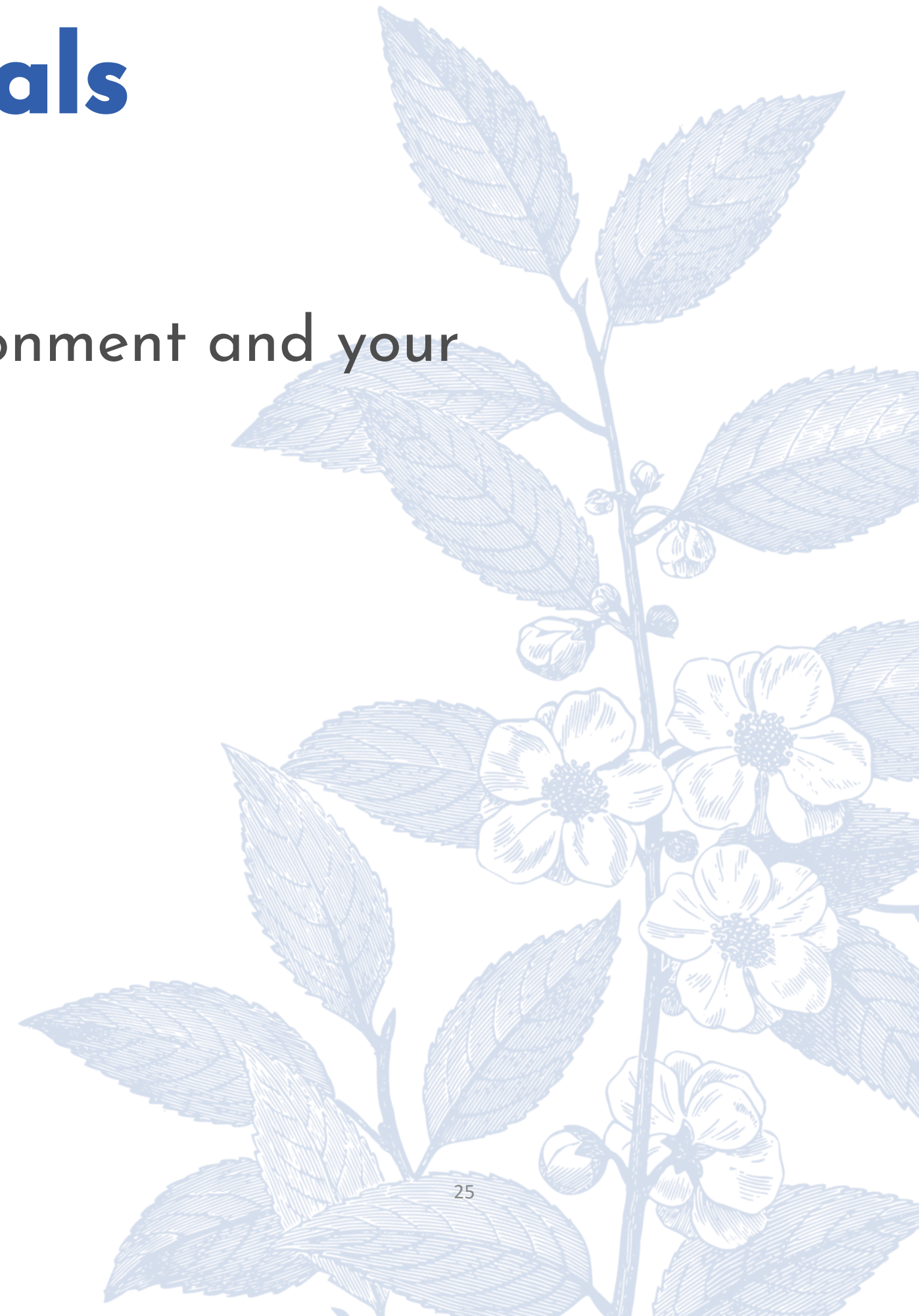
Identifiers

✓Identifier: name of a variable, method, or class

✓Rules for identifiers in Java:

- Can be made up of letters, digits, and the underscore (_) and dollar sign ($) characters
- Cannot start with a digit
- Cannot use other symbols such as ? or %
- Spaces are not permitted inside identifiers
- You cannot use reserved words such as public
- They are case sensitive

Identifiers

✓By convention, variable names start with a lowercase letter
"Camel case": Capitalize the first letter of a word in a compound word such as farewellMessage

✓By convention, class names start with an uppercase letter

✓Do not use the $ symbol in names – it is intended for names that are automatically generated by tools

Java Reserved Keywords

abstract	double	instanceof	static
assert	else	int	strictfp
boolean	enum	interface	super
break	extends	long	switch
byte	false	native	synchronized
case	for	new	this
catch	final	null	throw
char	finally	package	throws
class	float	private	transient
const	goto	protected	true
continue	if	public	try
default	implements	return	void
do	import	short	volatile
			while

Java Escape Sequences

\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
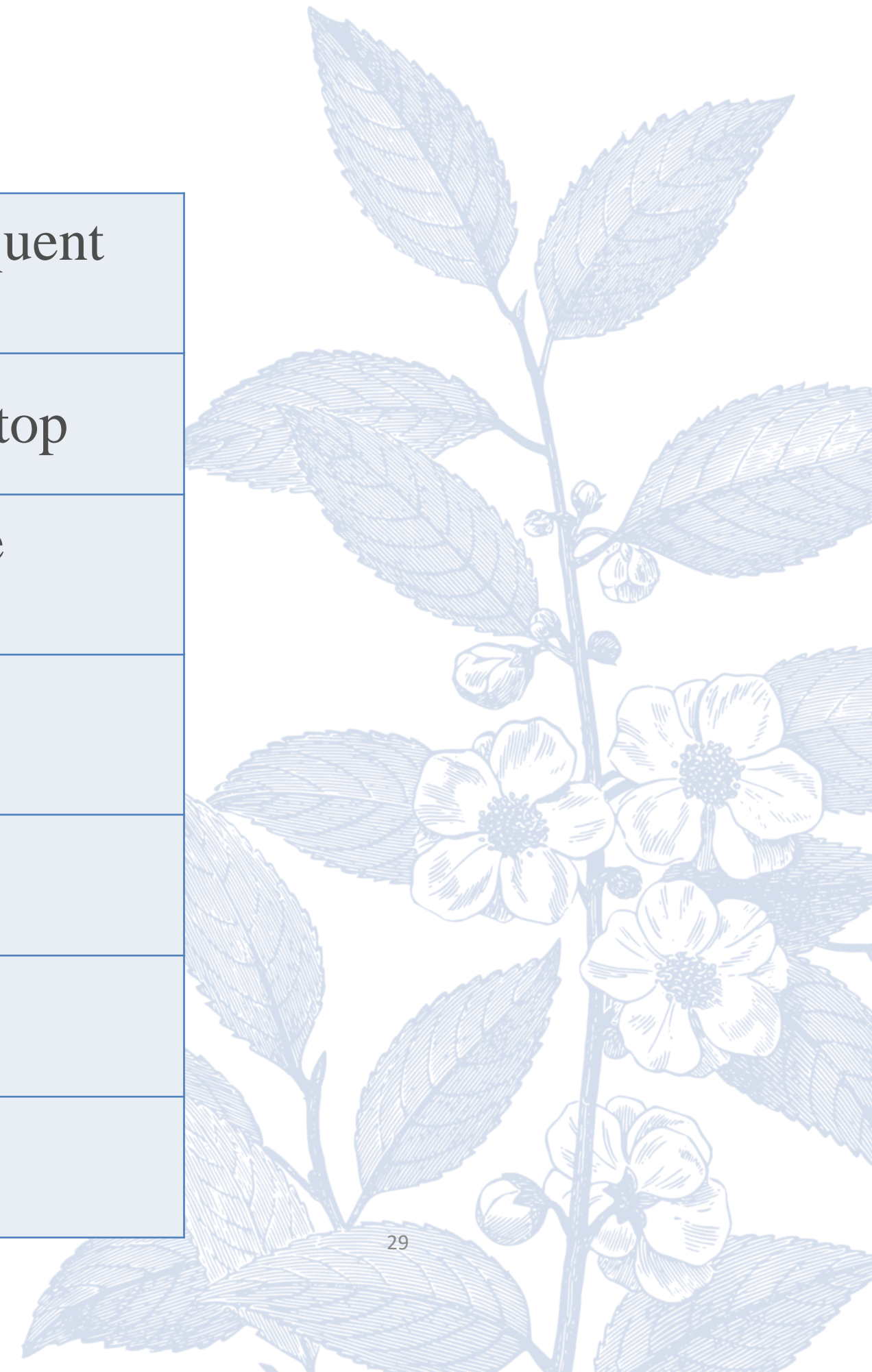
Java Naming Conventions

✓Variable names should begin with a lower case letter and then switch to title case thereafter:

- ▪Ex: int caTaxRate

✓Class names should be all title case.

- ▪Ex: public class BigLittle

✓More Java naming conventions can be found at: http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html

✓A general rule of thumb about naming variables and classes are that, with some exceptions, their names tend to be nouns or noun phrases.
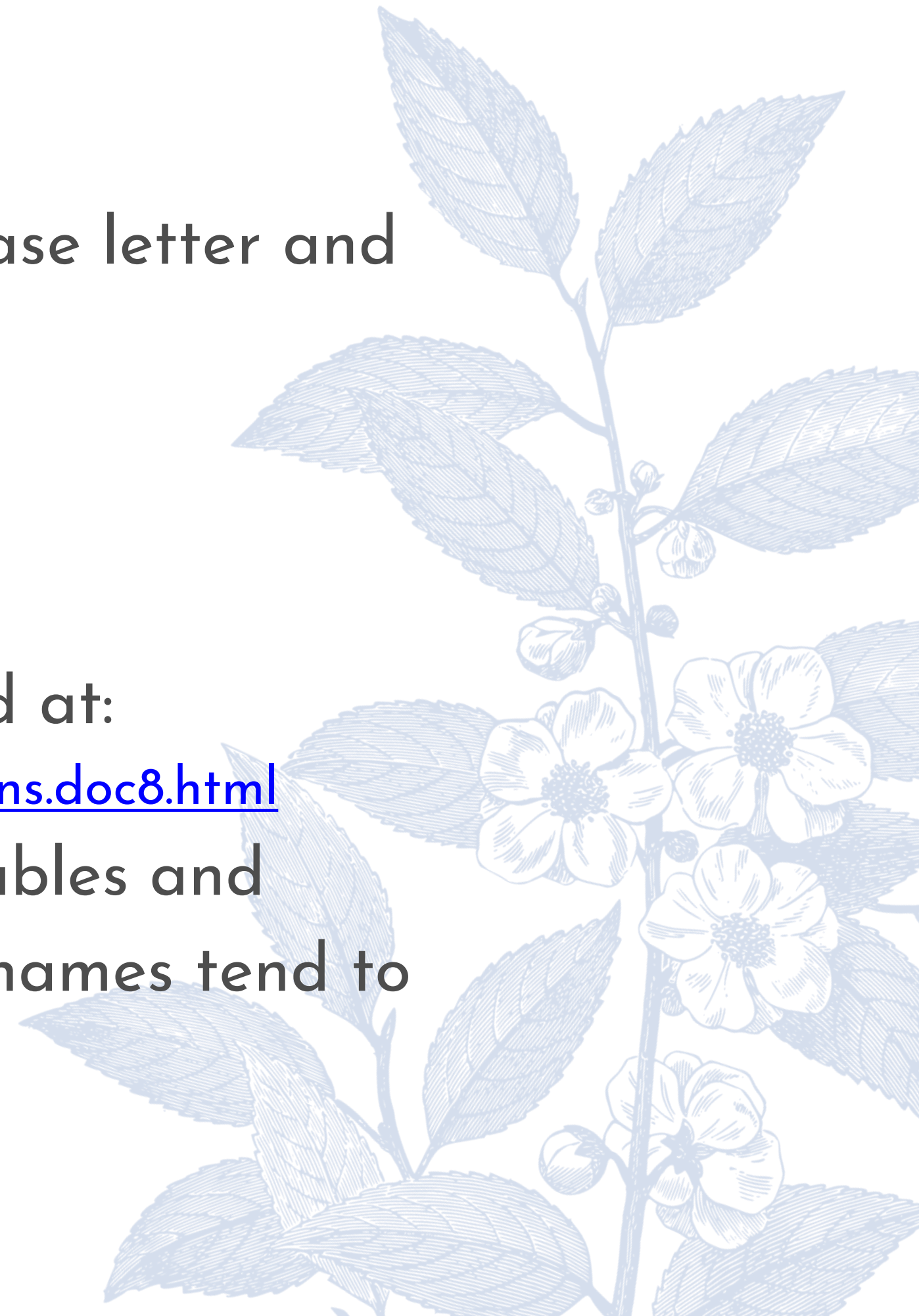
Comments

✓As your programs get more complex, you should add comments, explanations for human readers of your code.

✓It is a good practice to provide comments. This helps programmers who read your code understand your intent. In addition, you will find comments helpful when you review your own programs.

✓You use the // delimiter for short comments.

✓If you have a longer comment, enclose it between /* and */ delimiters.

✓The compiler ignores these delimiters and everything in between.

✓Ex:

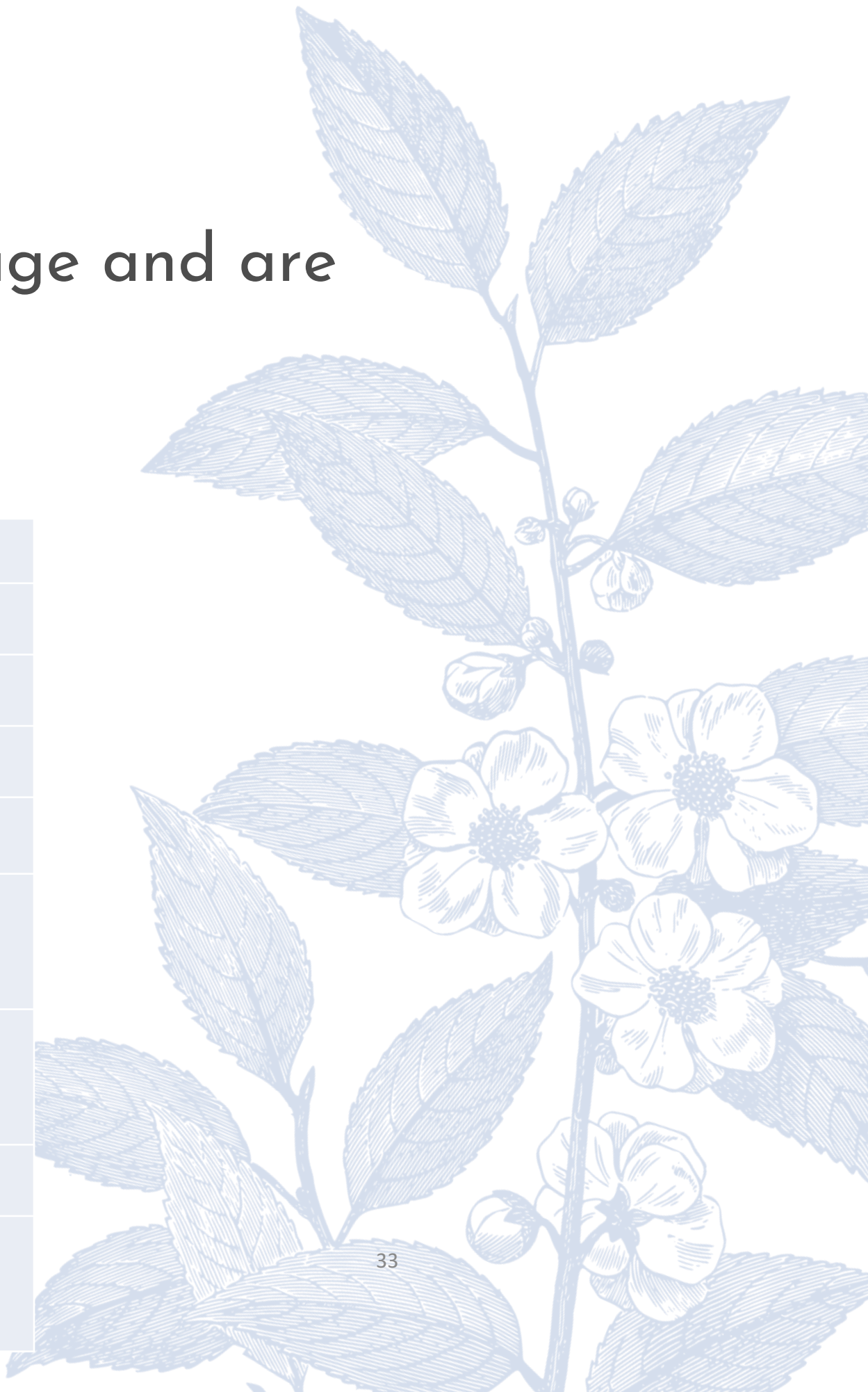double milesPerGallon = 35.5; // The average fuel efficiency of new U.S. cars in 2013

Types

- A type defines a set of values and the operations that can be carried out on the values
- Ex:
  - 13 has type int
  - "Hello, World" has type String
  - System.out has type PrintStream
- Java has separate types for integers and floating-point numbers
  - The double type denotes floating-point numbers
- A value such as 13 or 1.3 that occurs in a Java program is called a number literal

Primitive Data Types

✓Primitive data types are built into the Java language and are not derived from classes.

✓There are 8 Java primitive data types.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2^{31} to $2^{31}-1$
long	8 bytes	Stores whole numbers from -2^{63} to $2^{63}-1$
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	The char data type is a single 16-bit Unicode character.

Number Literals

Number	Type	Comment
6	int	An integer has no fractional part.
−6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	double	A number with a fractional part has type `double`.
1.0	double	An integer with a fractional part .0 has type `double`.
1E6	double	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type `double`.
2.96E-2	double	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
100000L	long	The `L` suffix indicates a long literal.
🚫 100,000		**Error:** Do not use a comma as a decimal separator.
100_000	int	You can use underscores in number literals.
🚫 3 1/2		**Error:** Do not use fractions; use decimal notation: 3.5

Variables

✓Use a variable to store a value that you want to use at a later time

✓A variable has a type, a name, and a value:

```
String greeting = "Hello, World!"
PrintStream printer = System.out;
int width = 13;
```

✓Variables can be used in place of the values that they store:

```
printer.println(greeting);
// Same as System.out.println("Hello, World!")
printer.println(width);
// Same asSystem.out.println(20)
```

Variables

✓It is an error to store a value whose type does not match the type of the variable:

  String greeting = 20; // ERROR: Types don't match

✓Syntax 2.1 Variable Declaration

Syntax   typeName variableName = value;
            or
            typeName variableName;

See Table 2 for rules and examples of valid names.

The type specifies what can be done with values stored in this variable.

String greeting = "Hello, Dave!";

A variable declaration ends with a semicolon.

Use a descriptive variable name.
See Programming Tip 2.1.

Supplying an initial value is optional, but it is usually a good idea.

Variable Declarations

Variable Name	Comment
`int width = 20;`	Declares an integer variable and initializes it with 20.
`int perimeter = 4 * width;`	The initial value need not be a fixed value. (Of course, `width` must have been previously declared.)
`String greeting = "Hi!";`	This variable has the type `String` and is initialized with the string "Hi".
🚫 `height = 30;`	**Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.2.5.
🚫 `int width = "20";`	**Error:** You cannot initialize a number with the string "20". (Note the quotation marks.)
`int width;`	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1.
`int width, height;`	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

Variable Names

Variable Name	Comment
distance_1	Names consist of letters, numbers, and the underscore character.
x	In mathematics, you use short variable names such as x or y. This is legal in Java, but not very common, because it can make programs harder to understand (see Programming Tip 2.1).
⚠ CanVolume	**Caution:** Names are case sensitive. This variable name is different from canVolume, and it violates the convention that variable names should start with a lowercase letter.
🚫 6pack	**Error:** Names cannot start with a number.
🚫 can volume	**Error:** Names cannot contain spaces.
🚫 double	**Error:** You cannot use a reserved word as a name.
🚫 miles/gal	**Error:** You cannot use symbols such as / in names.

Scope

✓Scope refers to the part of a program that has access to a variable's contents.

✓Variables declared inside a method (like the main method) are called local variables.

✓Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.

Operator Precedence

| Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR		
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

High

Low

The + Operator

✓The + operator can be used in two ways.

▪as a concatenation operator

▪as an addition operator

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

```
System.out.println("Hello " + "World");
System.out.println("The value is: " + 5);
System.out.println("The value is: " + value);
System.out.println("The value is: " + '/n' + 5);
```
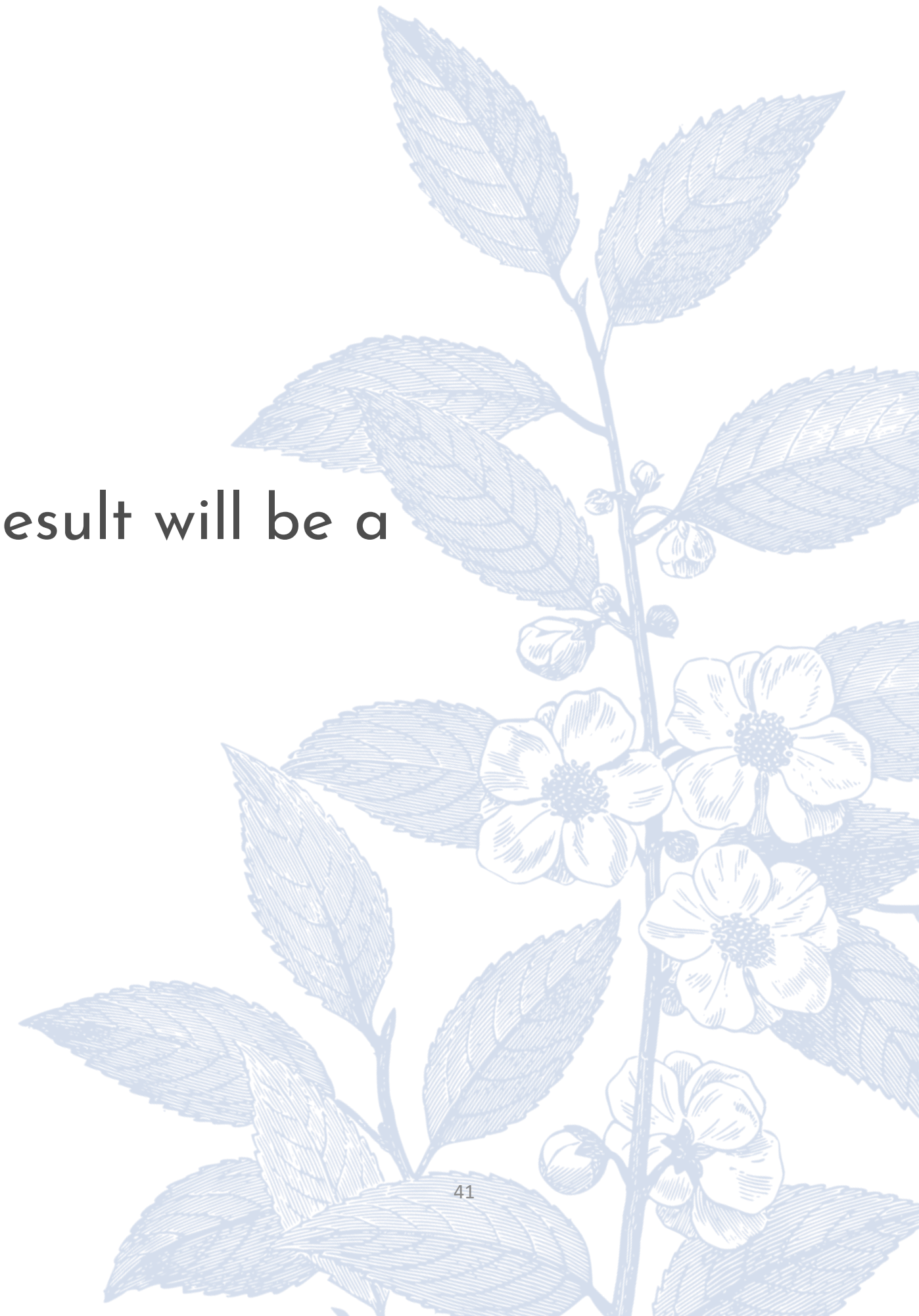
The Assignment Operator

Syntax 2.2 Assignment

Syntax	variableName = value;

```
                          double width = 20;
                              .
                              .
                          width = 30;
                              .
                              .
                              .
                          width = width + 10;
```

This is a variable declaration.

This is an assignment statement.

The value of this variable is changed.

The new value of the variable

The same name can occur on both sides. See Figure 4.
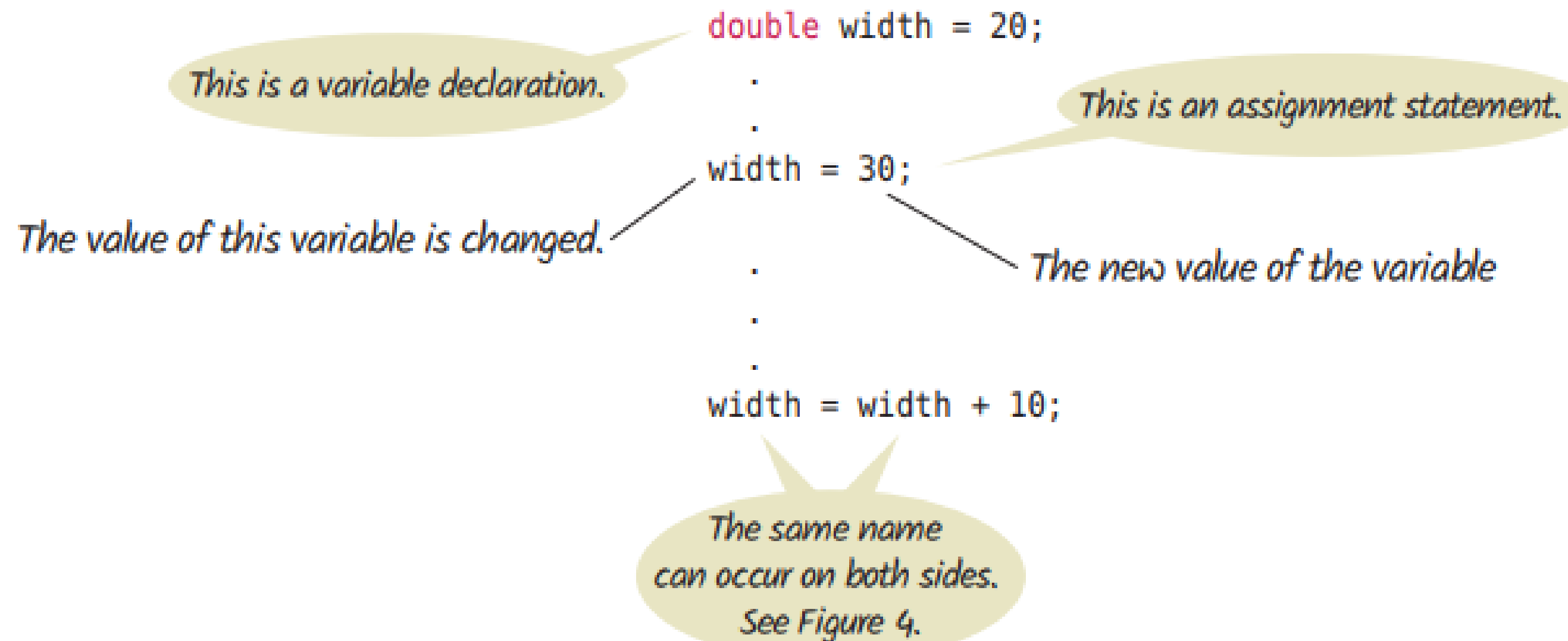
Variable Assignment and Initialization

✓In order to store a value in a variable, an *assignment statement* must be used.

✓The *assignment operator* is the equal (=) sign.

✓The operand on the left side of the assignment operator must be a variable name.

✓The operand on the right side must be either a literal or expression that evaluates to a type that is compatible with the type of the variable.
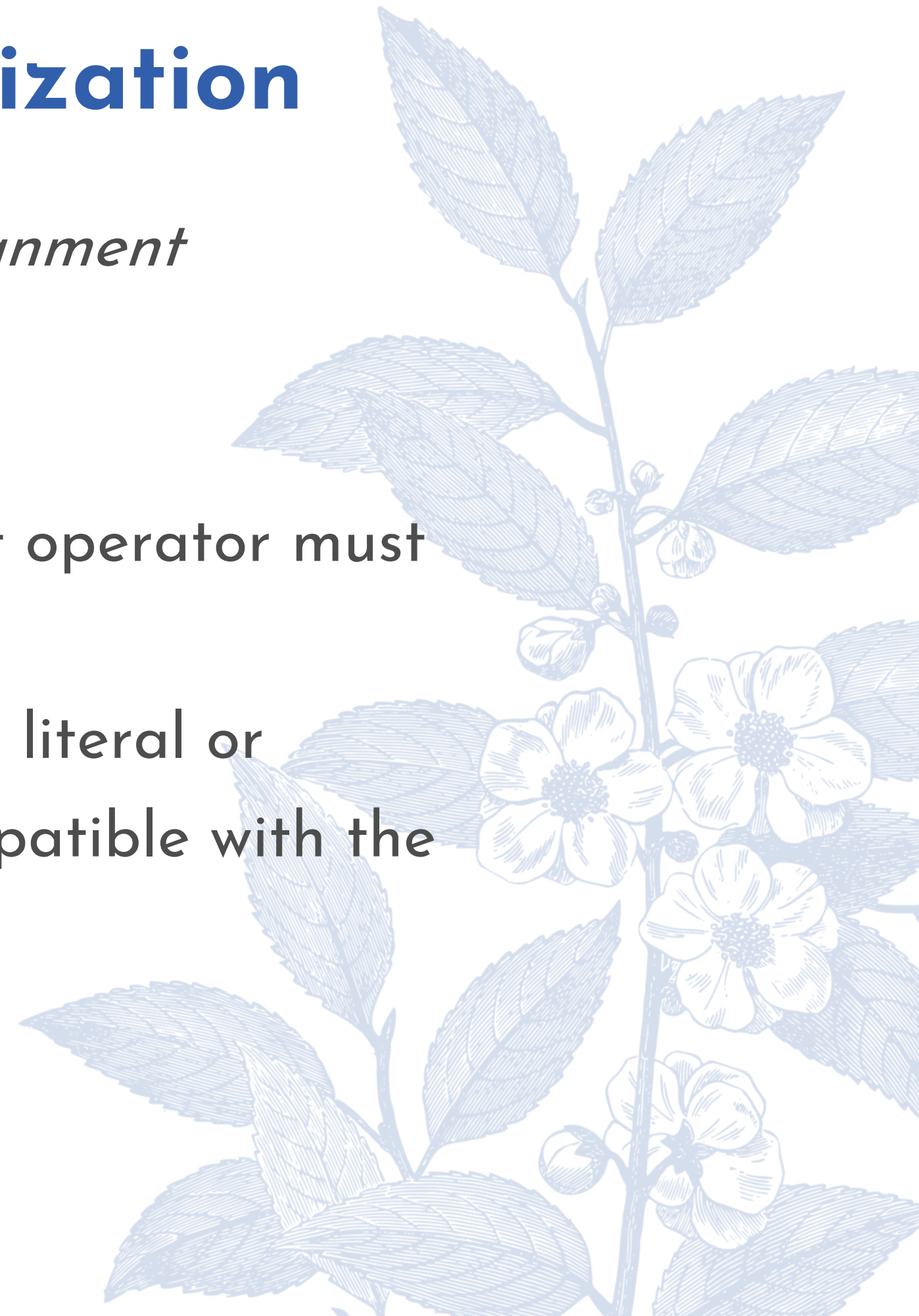
Variable Assignment and Initialization

✓It is an error to use a variable that has never had a value
  assigned to it:
    int height;
    width = height; // ERROR—uninitialized variable height
✓Remedy: assign a value to the variable before you use it:
    int height = 30;
    width = height; // OK
✓Even better, initialize the variable when you declare it:
    int height = 30;
    int width = height; // OK
✓The variables must be declared before they can be used.

Constants

✓Many programs have data that does not need to be changed.

✓Constants allow the programmer to use a name rather than a value throughout the program.

✓Constants keep the program organized and easier to maintain.

✓Constants are identifiers that can hold only a single value.

✓Constants are declared using the keyword final.

✓By convention, constants are all upper case and words are separated by the underscore character.

▪Ex: final int CAL_SALES_TAX = 0.725;

The String Class

✓Java has no primitive data type that holds a series of characters.

✓The String class from the Java standard library is used for this purpose.

✓In order to be useful, the a variable must be created to reference a String object.

  String str;

✓A variable can be assigned a String literal.

  String value = "Hello";

✓Strings are the only objects that can be created in this way. A variable can be created using the new keyword.

  String value = new String("Hello");

Enumerated Types

✓Java allows you to define an enumerated type, which can then be used to declare variables

✓An enumerated type declaration lists all possible values for a variable of that type

✓The values are identifiers of your own choosing

✓The following declaration creates an enumerated type called Season

  Ex: enum Season {WINTER, SPRING, SUMMER, FALL};

Enumerated Types
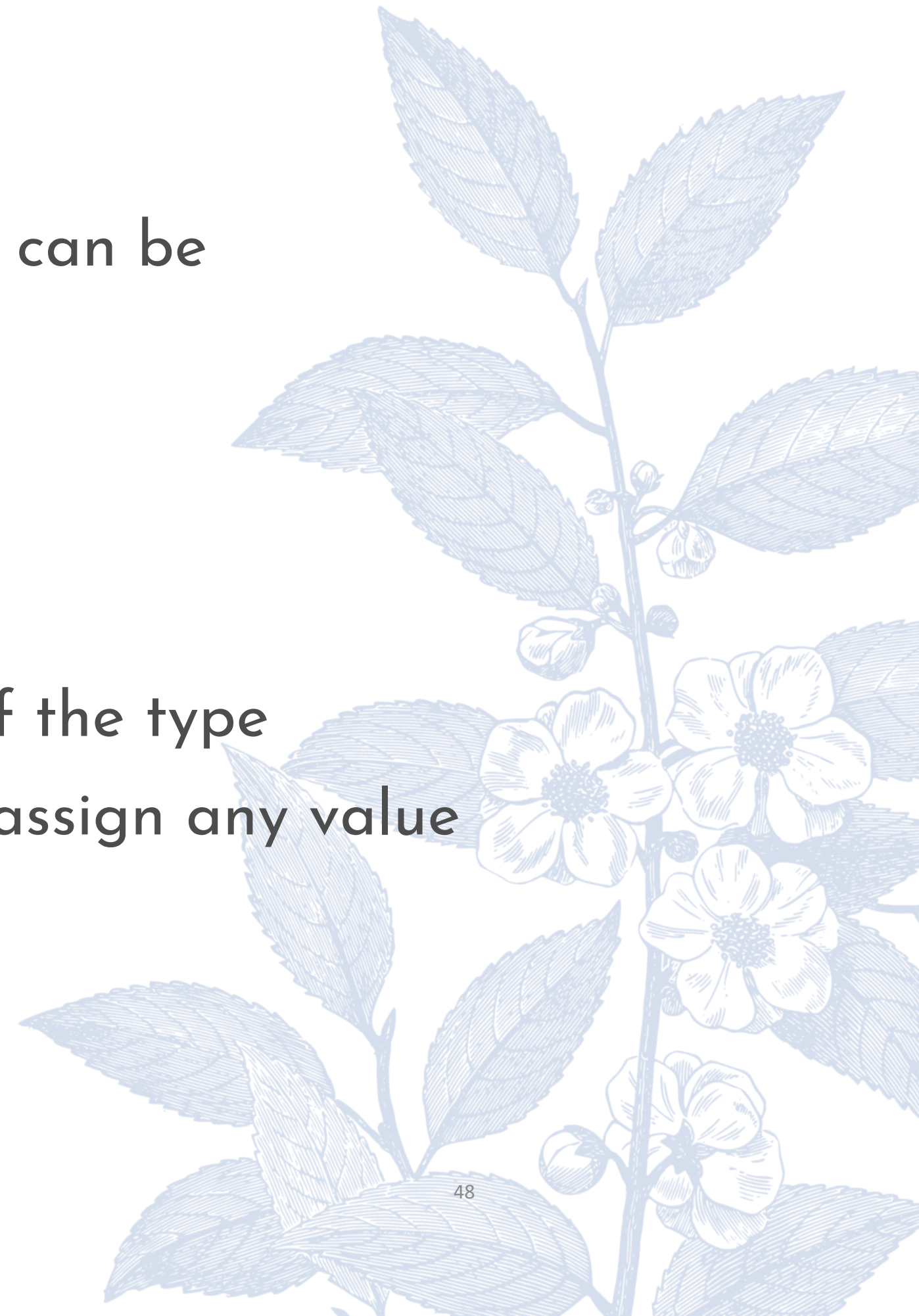
✓Once a type is defined, a variable of that type can be declared:

 Season time;

✓And it can be assigned a value:

 time = Season.FALL;

✓The values are referenced through the name of the type

✓Enumerated types are type-safe – you cannot assign any value other than those listed

Ordinal Values

✓Internally, each value of an enumerated type is stored as an integer, called its ordinal value

✓The first value in an enumerated type has an ordinal value of zero, the second one, and so on

✓However, you cannot assign a numeric value to an enumerated type, even if it corresponds to a valid ordinal value

✓The declaration of an enumerated type is a special type of class, and each variable of that type is an object

Wrapper Classes

✓The java.lang package contains wrapper classes that correspond to each primitive type:

Primitive Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean
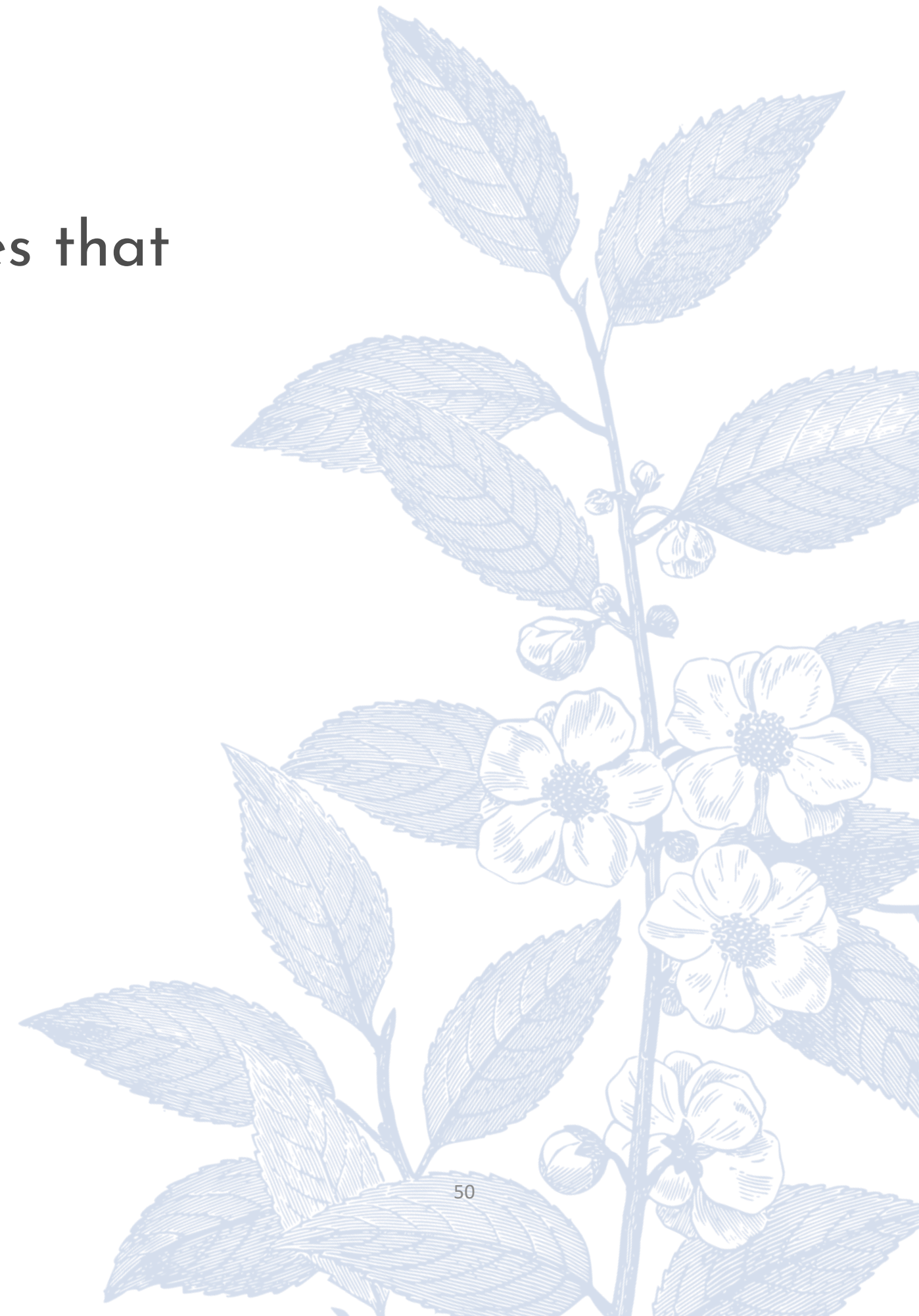
Wrapper Classes

✓The following declaration creates an Integer object which represents the integer 40 as an object

   Integer age = new Integer(40);

✓An object of a wrapper class can be used in any situation where a primitive value will not suffice

✓Wrapper classes also contain static methods that help manage the associated type

   For example, the Integer class contains a method to convert an integer stored in a String to an int value:

   num = Integer.parseInt(str);

✓They often contain useful constants as well

   For example, the Integer class contains MIN_VALUE and MAX_VALUE which hold the smallest and largest int values

Autoboxing

✓Autoboxing is the automatic conversion of a primitive value to a corresponding wrapper object:

    Integer obj;

    int num = 42;

    obj = num;

✓The assignment creates the appropriate Integer object

✓The reverse conversion (called unboxing) also occurs automatically as needed

    Character ch = new Character('T');

    char myChar = ch;

    The char in the object is unboxed before the assignment.
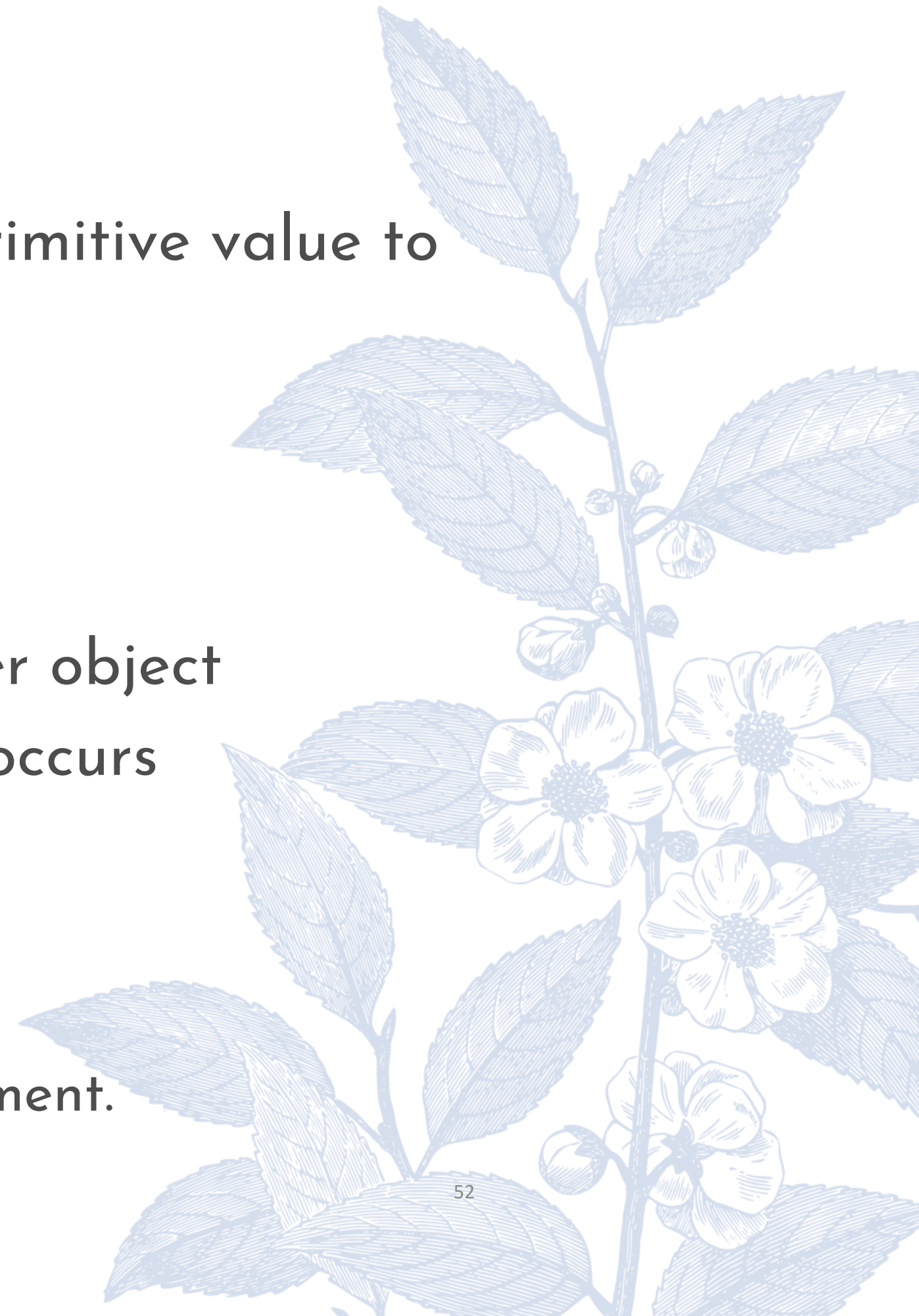
Packages

✓For purposes of accessing them, classes in the Java API are organized into packages

✓These often overlap with specific APIs

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities

The import Declaration

✓When you want to use a class from a package, you could use

  its fully qualified name

    java.util.Scanner

✓Or you can import the class, and then use just the class name

    import java.util.Scanner;

✓To import all classes in a particular package, you can use the *

  wildcard character

    import java.util.*;

✓All classes of the java.lang package are imported

  automatically into all programs

Chapter 1 - Java Fundamentals

Chapter Goals

✓To become familiar with your computing environment and your compiler

✓To compile and run your first Java program

✓To recognize syntax and logic errors

✓To write pseudocode for simple algorithms

✓To learn about Data and Expressions

✓To learn about Predefined classes

✓To learn about Conditionals and Loops

✓To learn about Arrays
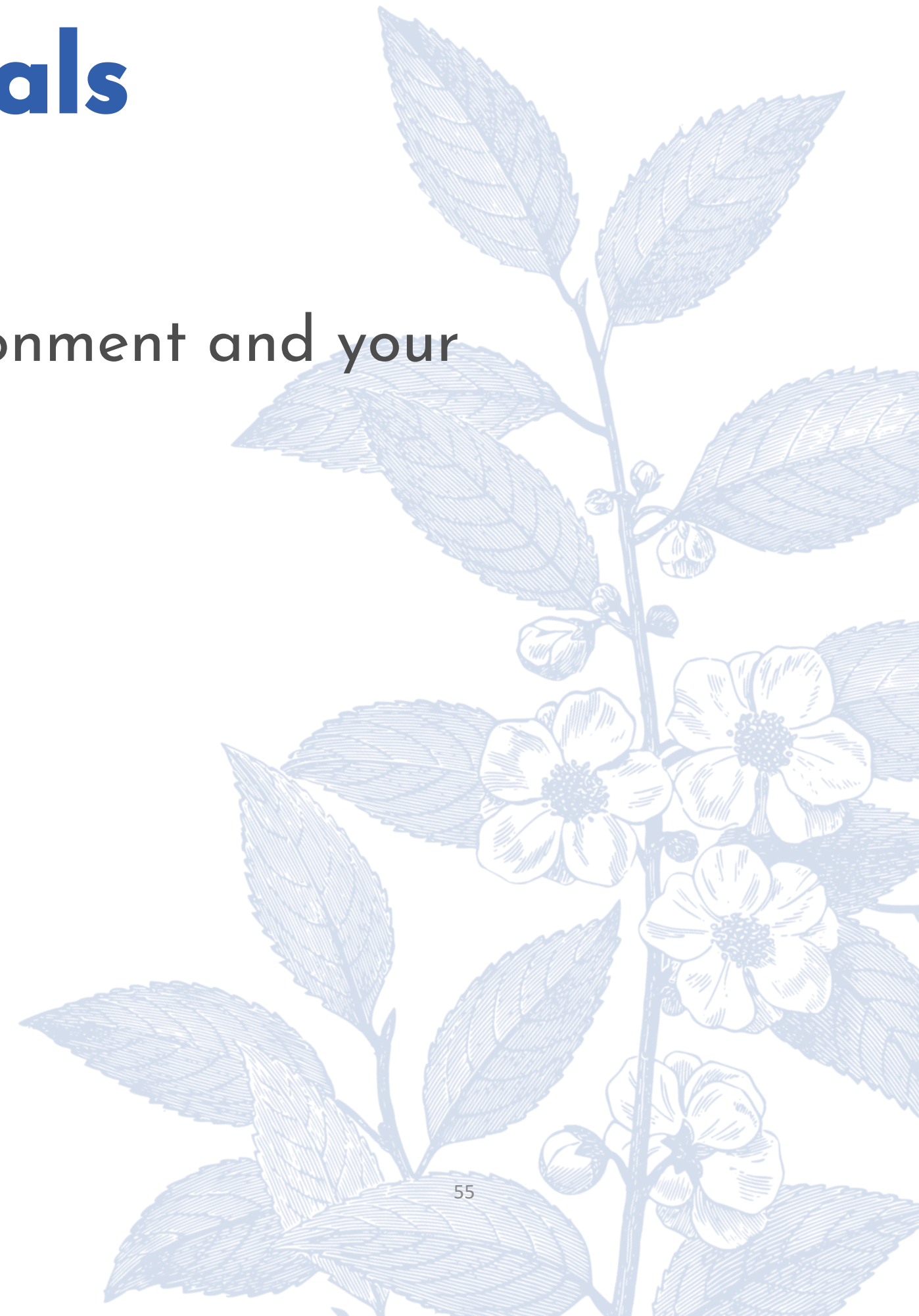
The Scanner Class

✓To read input from the keyboard we can use the Scanner class.

✓The Scanner class is defined in java.util, so we will use the following statement at the top of our programs:

import java.util.Scanner;

✓Scanner objects work with System.in

✓To create a Scanner object:

Scanner keyboard = new Scanner (System.in);

✓Scanner class methods: nextInt(), nextFloat()...

The Random Class

✓The Random class is part of the java.util package

✓It provides methods that generate pseudorandom numbers

✓A Random object performs complicated calculations based on a seed value to produce a stream of seemingly random values

✓Ex:

	Range
gen.nextInt(25)	0 to 24
gen.nextInt(6) + 1	1 to 6
gen.nextInt(100) + 10	10 to 109
gen.nextInt(50) + 100	100 to 149
gen.nextInt(10) – 5	-5 to 4
gen.nextInt(22) + 12	12 to 33

The Math Class

✓The Math class is part of the java.lang package

✓The Math class contains methods that perform various mathematical functions

✓These include:
  ▪absolute value
  ▪square root
  ▪exponentiation
  ▪trigonometric functions

✓The methods of the Math class are static methods

Formatting Output

✓It is often necessary to format output values in certain ways so that they can be presented properly

✓The Java standard class library contains classes that provide formatting capabilities

✓The NumberFormat class allows you to format values as currency or percentages

✓The DecimalFormat class allows you to format values based on a pattern

✓Both are part of the java.text package

Formatting Output

✓The NumberFormat class has static methods that return a formatter object
  getCurrencyInstance()
  getPercentInstance()

✓Each formatter object has a method called format that returns a string with the specified information in the appropriate format

✓ The DecimalFormat class can be used to format a floating point value in various ways

✓For example, you can specify that the number should be truncated to three decimal places

✓The constructor of the DecimalFormat class takes a string that represents a pattern for the formatted number

Chapter 1 - Java Fundamentals

Chapter Goals

✓To become familiar with your computing environment and your compiler

✓To compile and run your first Java program

✓To recognize syntax and logic errors

✓To write pseudocode for simple algorithms

✓To learn about Data and Expressions

✓To learn about Predefined classes

✓To learn about Conditionals and Loops

✓To learn about Arrays
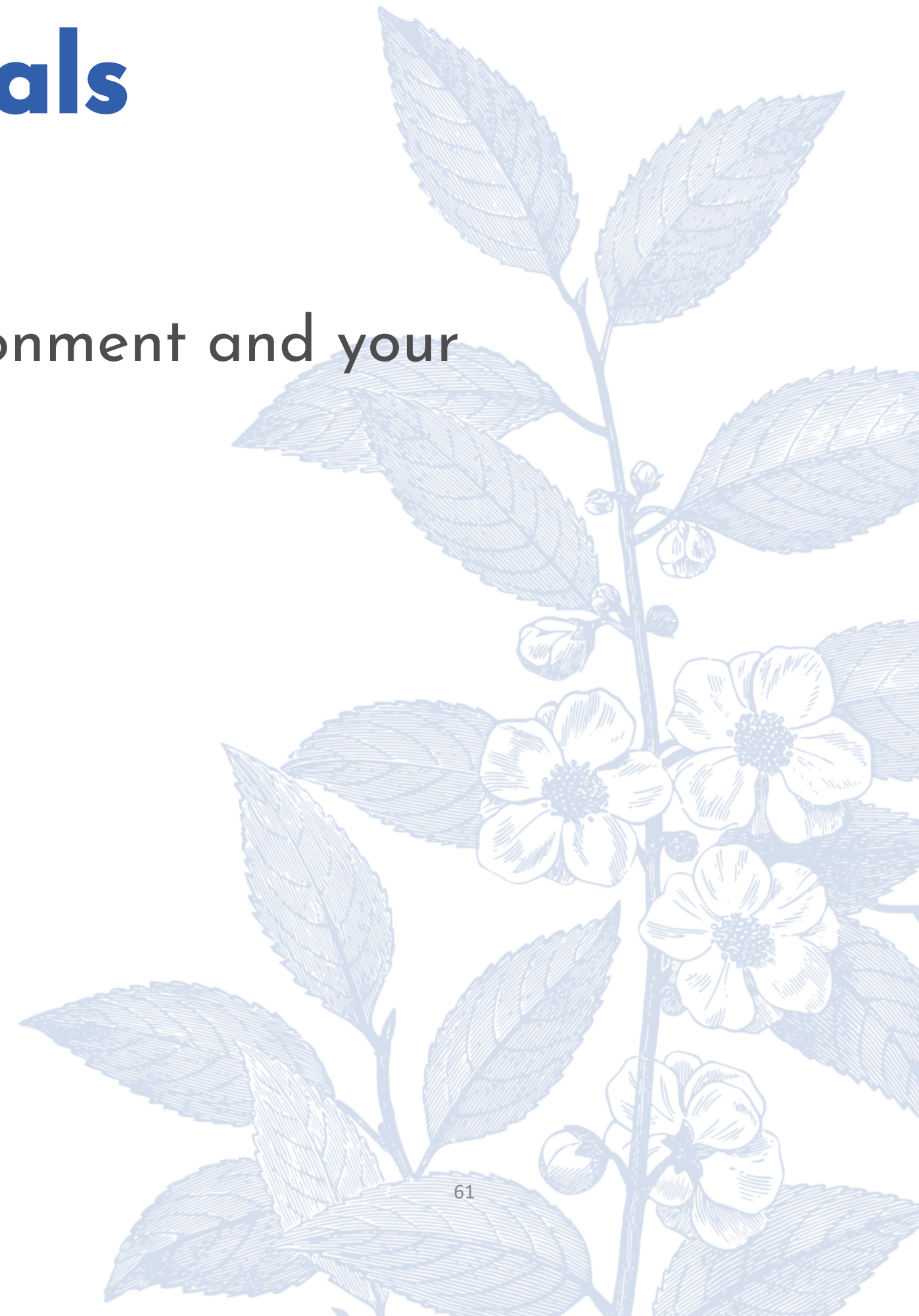
Flow of Control

✓Unless specified otherwise, the order of statement execution through a method is linear: one after another

✓Some programming statements allow us to make decisions and perform repetitions

✓These decisions are based on boolean expressions (also called conditions) that evaluate to true or false

✓The order of statement execution is called the flow of control

Conditional Statements

✓A conditional statement lets us choose which statement will be executed next

✓They are sometimes called selection statements

✓Conditional statements give us the power to make basic decisions

✓The Java conditional statements are the:
- if and if-else statement
- switch statement

✓A condition often uses one of Java's equality operators or relational operators

Logical Operators

✓Boolean expressions can also use the following logical operators:

!   Logical NOT

&& Logical AND

|| Logical OR

✓They all take boolean operands and produce boolean results

✓Logical NOT is a unary operator (it operates on one operand)

✓Logical AND and logical OR are binary operators (each operates on two operands)

The if Statement

✓Let's now look at the if statement in more detail

✓The if statement has the following syntax:

    if (condition)

      statement;

✓An else clause can be added to an if statement to make an if-else statement
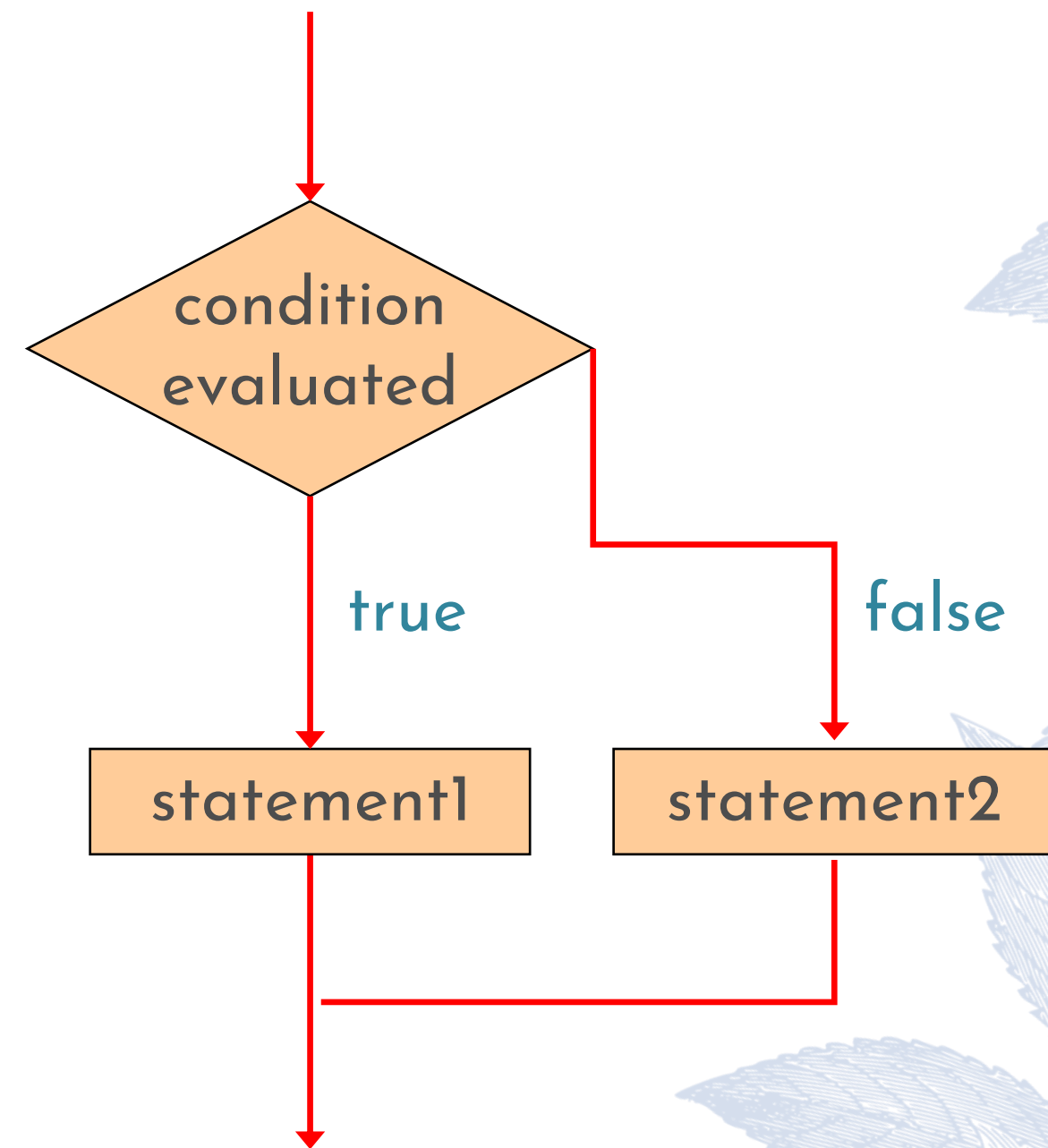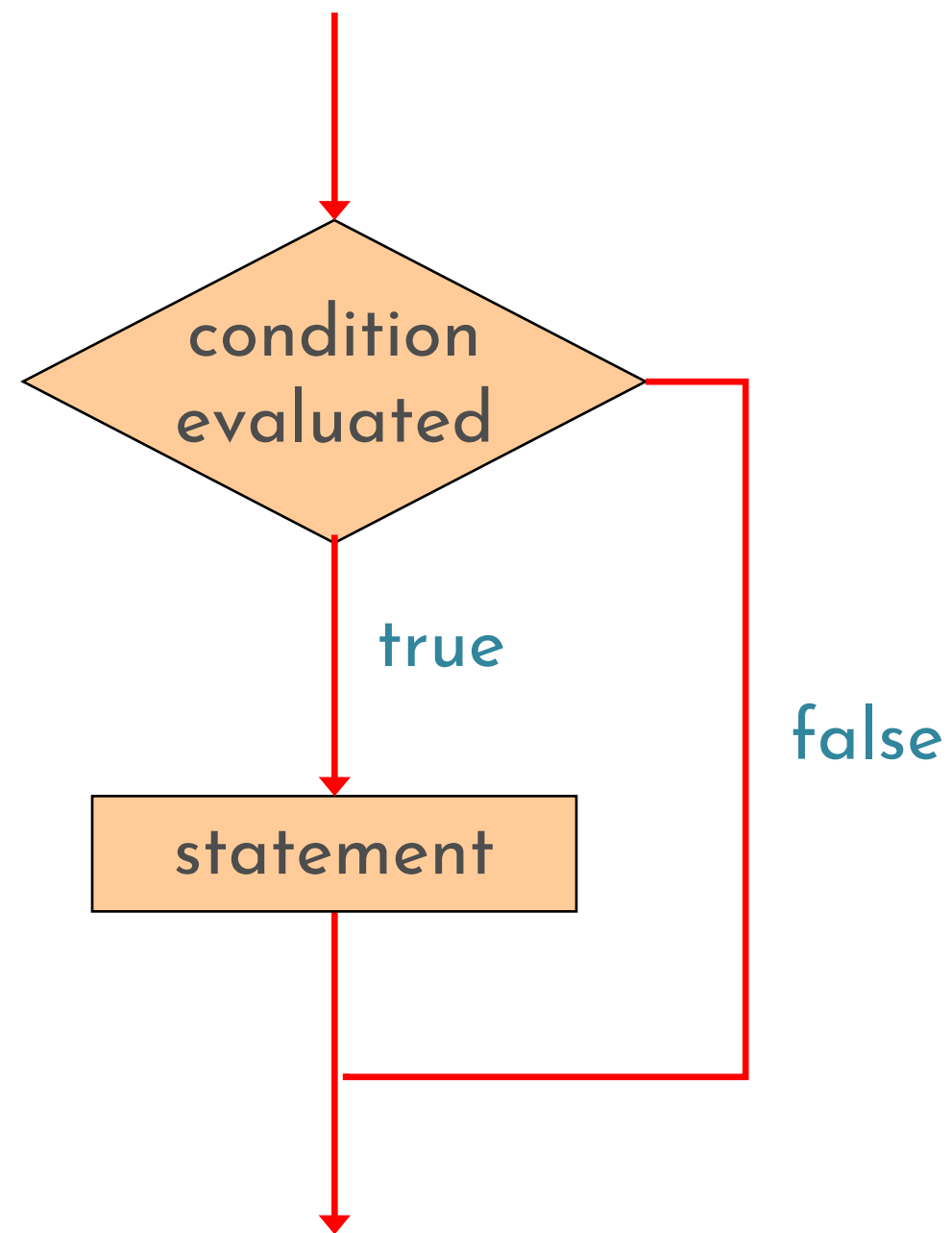
    if (condition)

      statement1;

    else

      statement2;

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

▪One or the other will be executed, but not both

Logic of an if statement

Nested if Statements

✓The statement executed as a result of an if or else clause could be another if statement

✓These are called nested if statements

✓An else clause is matched to the last unmatched if (no matter what the indentation implies)

✓Braces can be used to specify the if statement to which an else clause belongs

Block Statements

✓Several statements can be grouped together into a block statement delimited by braces

✓A block statement can be used wherever a statement is called for in the Java syntax rules

✓Ex:
```
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
```

The switch Statement

✓The if-else statement allows you to make true / false branches.

✓The switch statement allows you to use an ordinal value to determine how a program will branch.

✓The switch statement takes the form:

```
switch (SwitchExpression){
  case CaseExpression:
    // place one or more statements here
    break;
  case CaseExpression:
    // place one or more statements here
    break;
    // case statements may be repeated
  default:
    // place one or more statements here
}
```
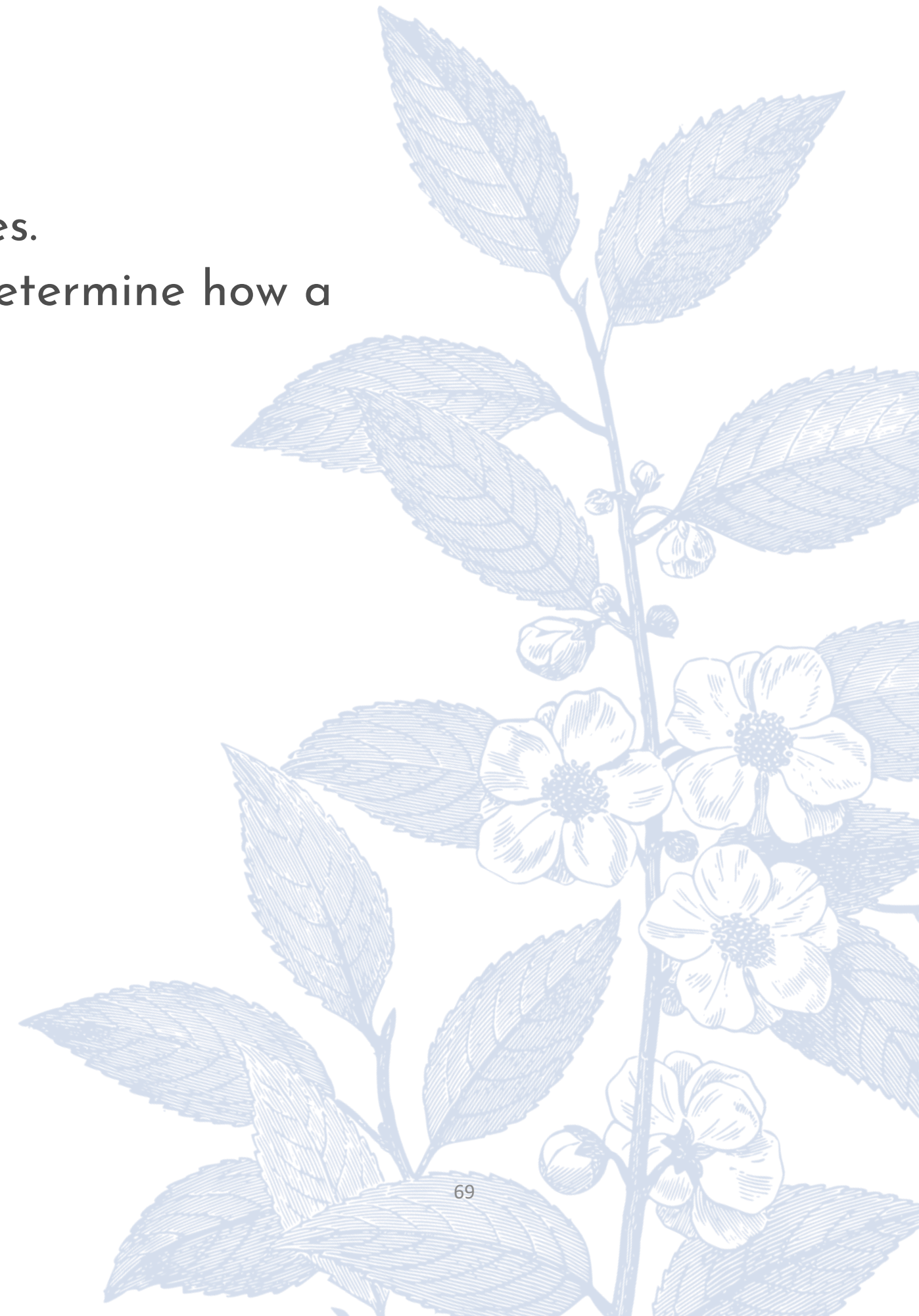
The Conditional Operator

✓The conditional operator evaluates to one of two expressions based on a boolean condition

✓Its syntax is:

  condition ? expression1 : expression2

✓If the condition is true, expression1 is evaluated; if it is false, expression2 is evaluated

✓The value of the entire conditional operator is the value of the selected expression

✓The conditional operator is ternary because it requires three operands

Comparing Data

✓When comparing data using boolean expressions, it's important to understand the nuances of certain data types

✓Let's examine some key situations:

- Comparing floating point values for equality
- Comparing characters
- Comparing strings (alphabetical order)
- Comparing object vs. comparing object references

Comparing Float Values

✓You should rarely use the equality operator (==) when comparing two floating point values (float or double)

✓Two floating point values are equal only if their underlying binary representations match exactly

✓Computations often result in slight differences that may be irrelevant

✓In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal

Comparing Float Values

✓To determine the equality of two floats, use the following

technique:

if (Math.abs(f1 - f2) < TOLERANCE)

System.out.println ("Essentially equal");

✓If the difference between the two floating point values is less

than the tolerance, they are considered to be equal

✓The tolerance could be set to any appropriate level, such as

0.000001

Comparing Characters

✓As we've discussed, Java character data is based on the Unicode character set

✓Unicode establishes a particular numeric value for each character, and therefore an ordering

✓We can use relational operators on character data based on this ordering

✓For example, the character '+' is less than the character 'J' because it comes before it in the Unicode character set

✓Appendix C provides an overview of Unicode

Comparing Characters

✓In Unicode, the digit characters (0-9) are contiguous and in order

✓Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122
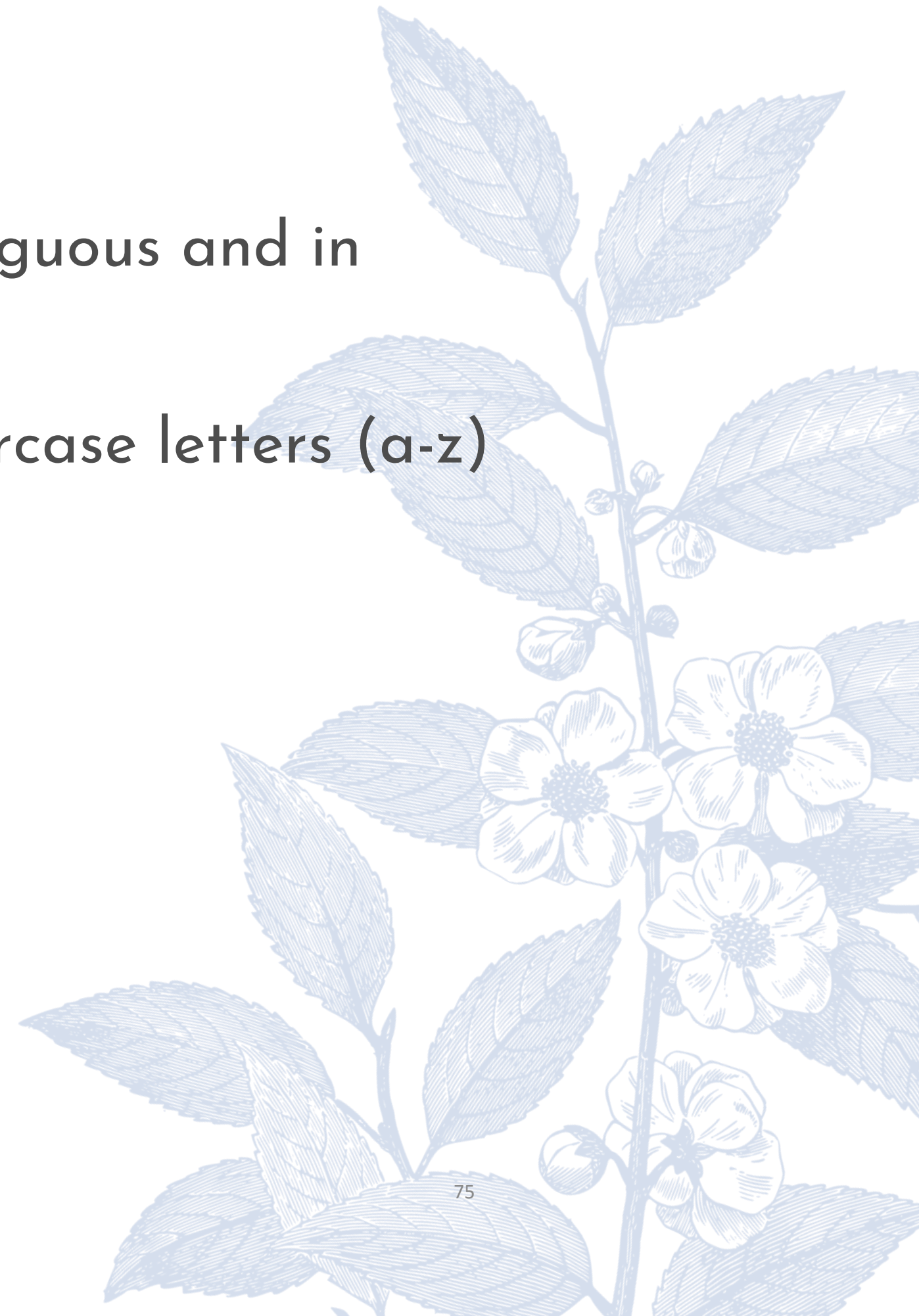
Comparing Strings

✓Remember that in Java a character string is an object

✓The equals method can be called with strings to determine if two strings contain exactly the same characters in the same order

✓The equals method returns a boolean result

```
if (name1.equals(name2))
    System.out.println ("Same name");
```

✓We cannot use the relational operators to compare strings

Comparing Strings

✓We cannot use the relational operators to compare strings

✓The String class contains the compareTo method for determining if one string comes before another

✓A call to name1.compareTo(name2)

- returns zero if name1 and name2 are equal (contain the same characters)
- returns a negative value if name1 is less than name2
- returns a positive value if name1 is greater than name2

Comparing Objects

✓The == operator can be applied to objects – it returns true if the two references are aliases of each other

✓The equals method is defined for all objects, but unless we redefine it when we write a class, it has the same semantics as the == operator

✓It has been redefined in the String class to compare the characters in the two strings

✓When you write a class, you can redefine the equals method to return true under whatever conditions are appropriate

Repetition Statements

✓Repetition statements allow us to execute a statement multiple times

✓Often they are referred to as loops

✓Like conditional statements, they are controlled by boolean expressions

✓Java has three kinds of repetition statements: while, do, and for loops

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, and if it is still true, the statement is executed again

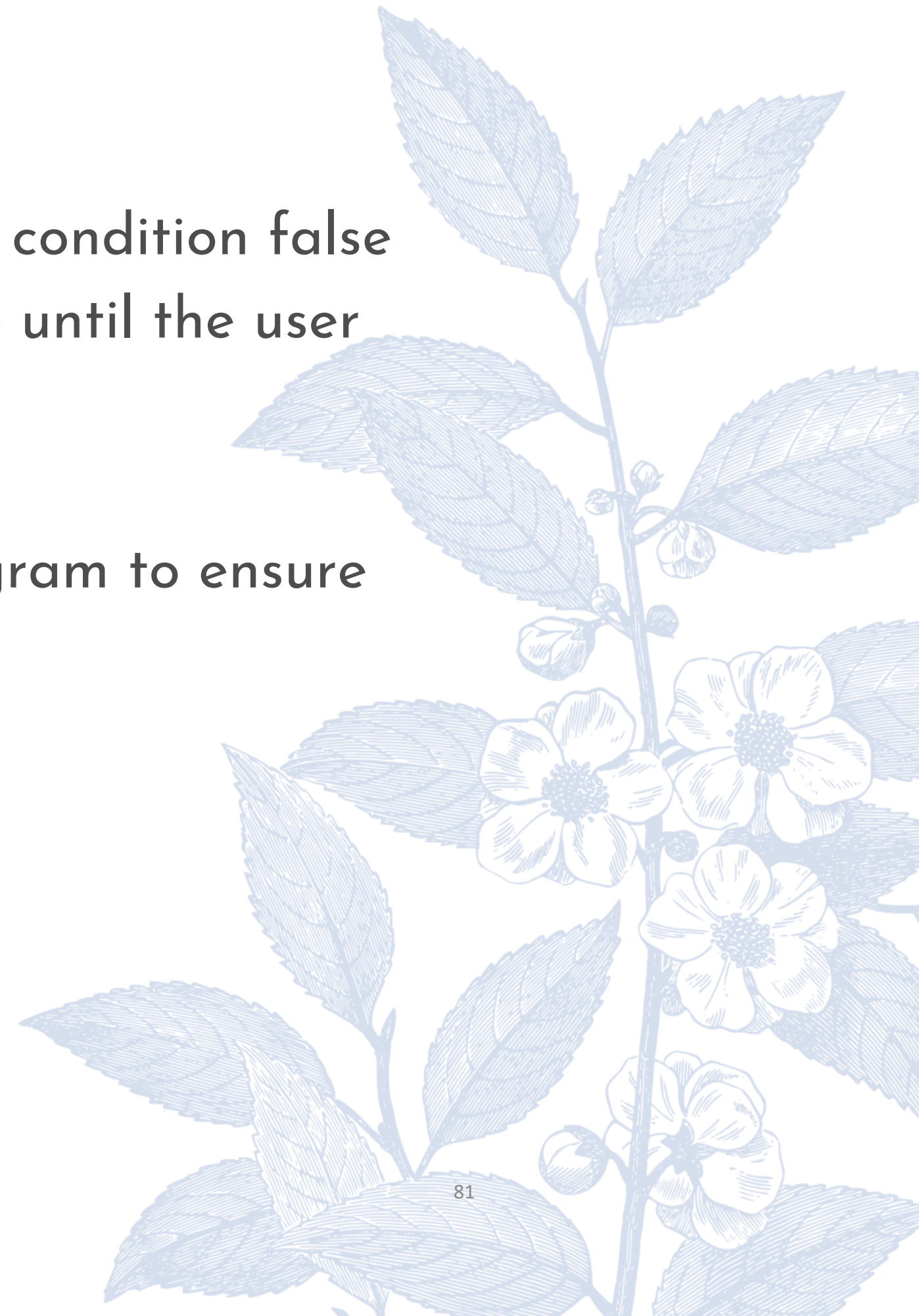✓The statement is executed repeatedly until the condition becomes false

condition evaluated

false

true

statement
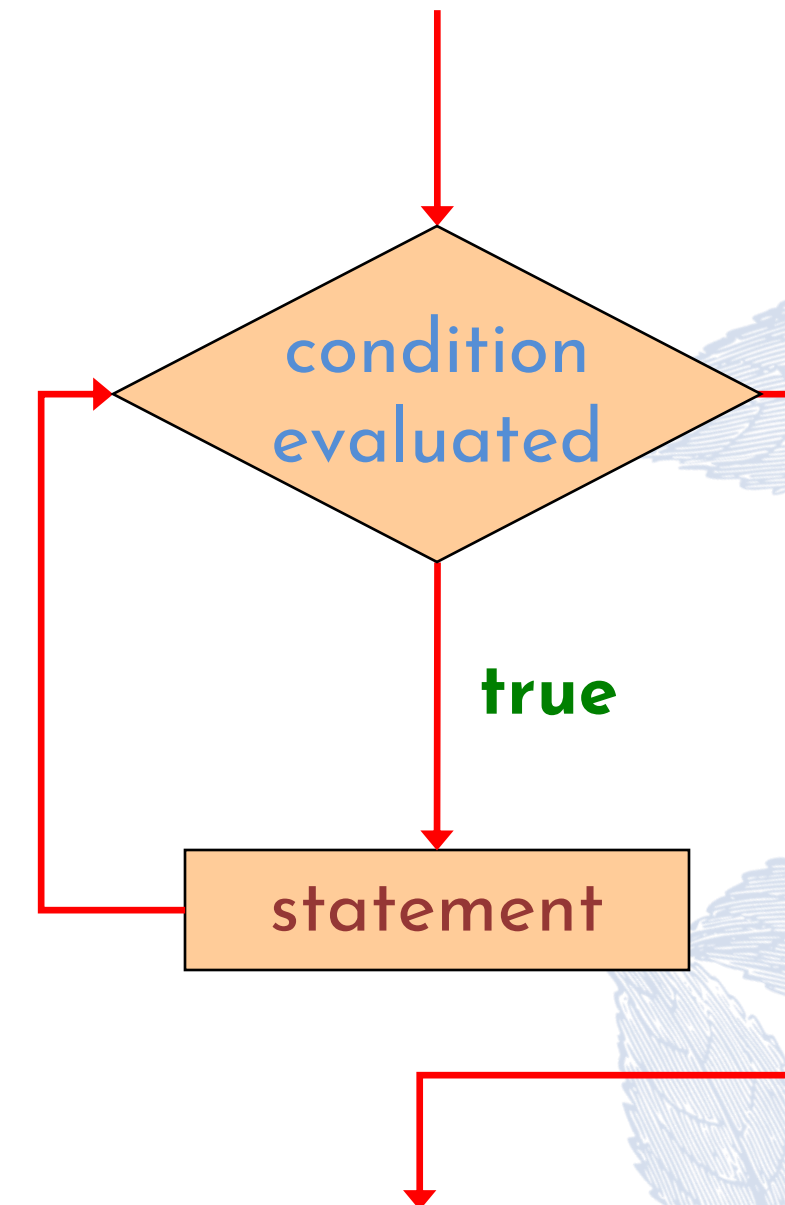
Infinite Loops

✓The body of a while loop eventually must make the condition false

✓If not, it is called an infinite loop, which will execute until the user interrupts the program

✓This is a common logical error

✓You should always double check the logic of a program to ensure that your loops will terminate normally

✓Ex:

```
int count = 1;
while (count <= 25){
    System.out.println (count);
    count = count - 1;
}
```
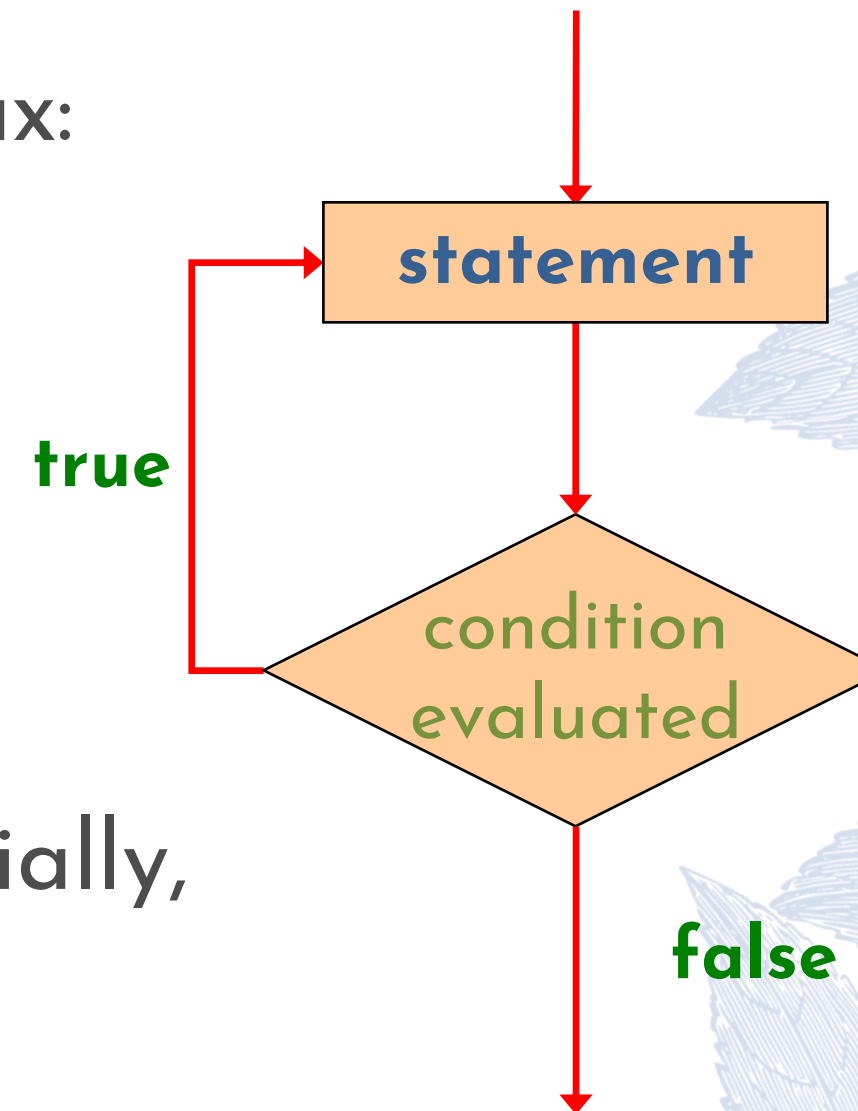
The do Statement

✓A do statement has the following syntax:
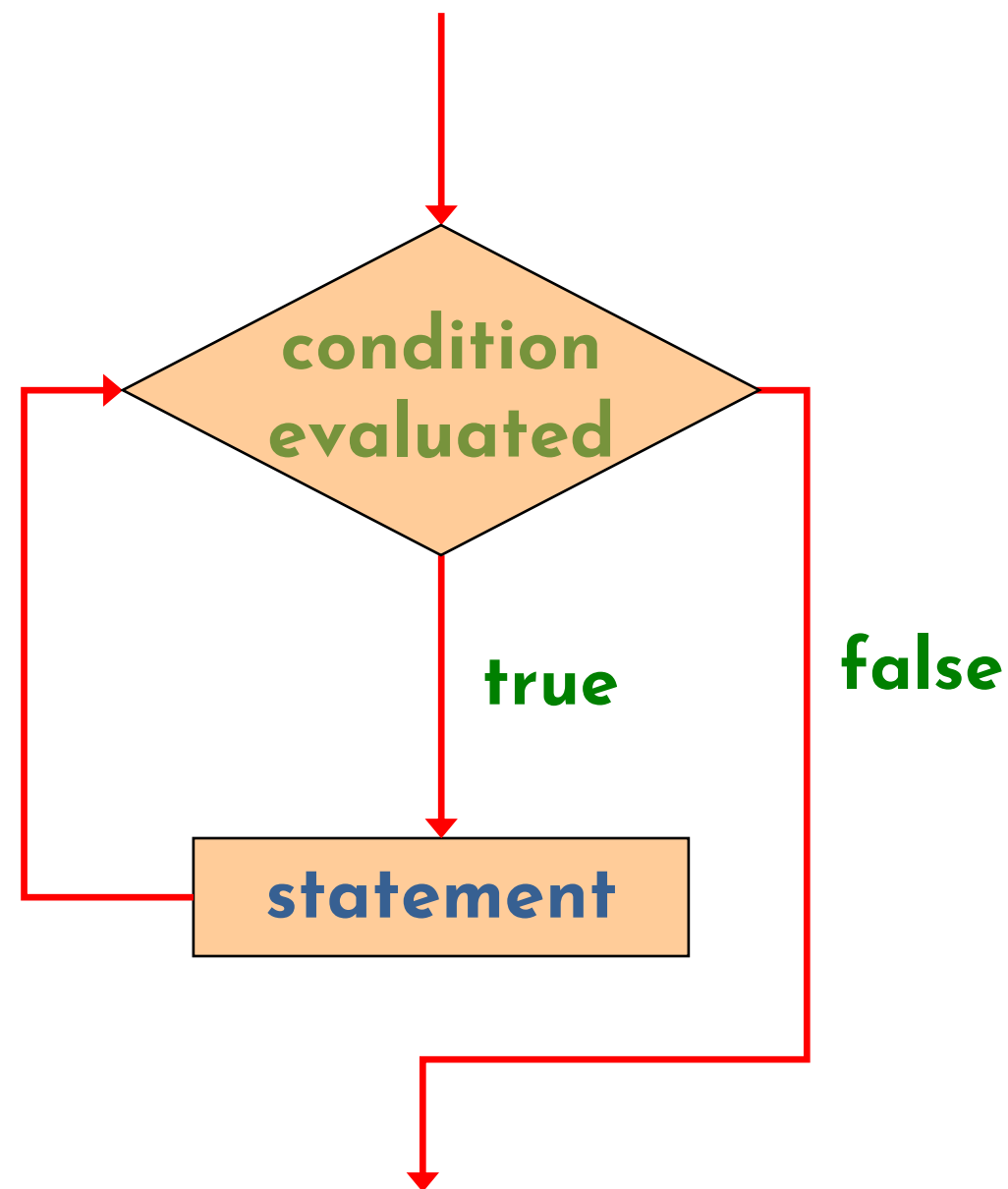
  do

  {

    statement-list;

  } while (condition);

✓The statement-list is executed once initially, and then the condition is evaluated

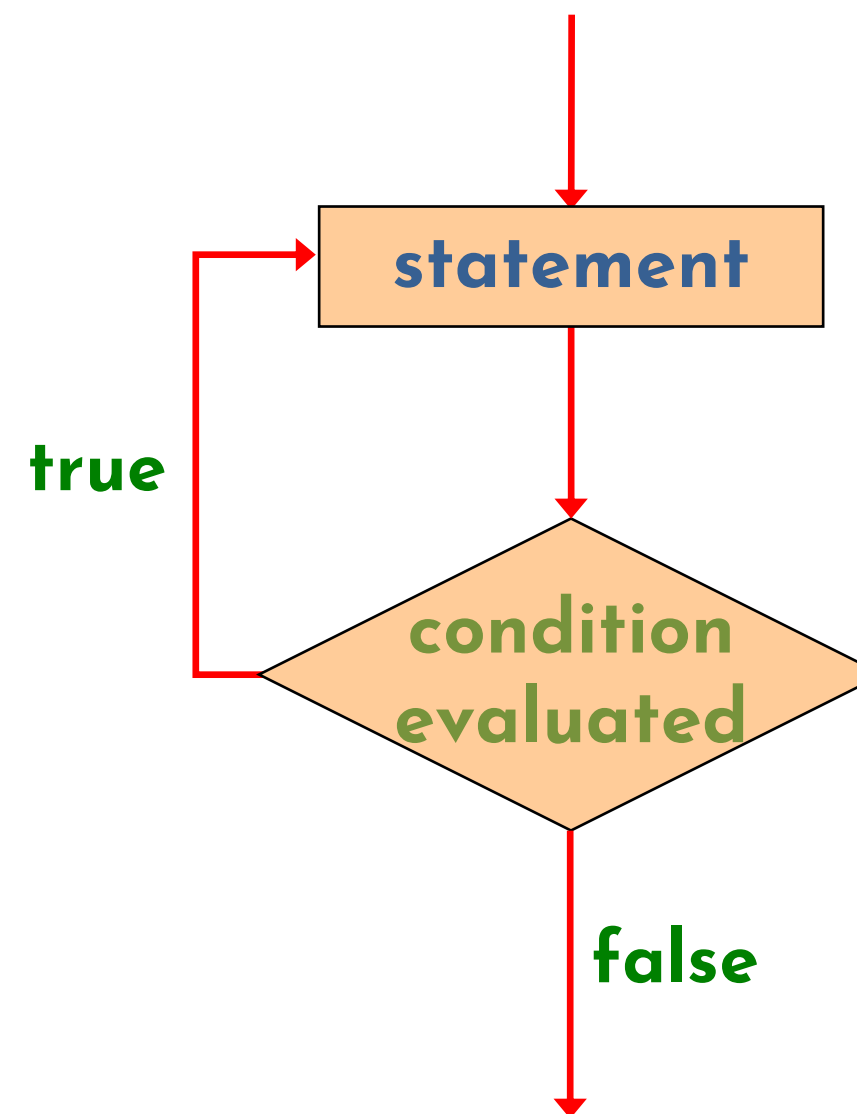✓The statement is executed repeatedly until the condition becomes false

statement

true

condition evaluated

false

Comparing while and do

The while Loop

The do Loop

The for Statement

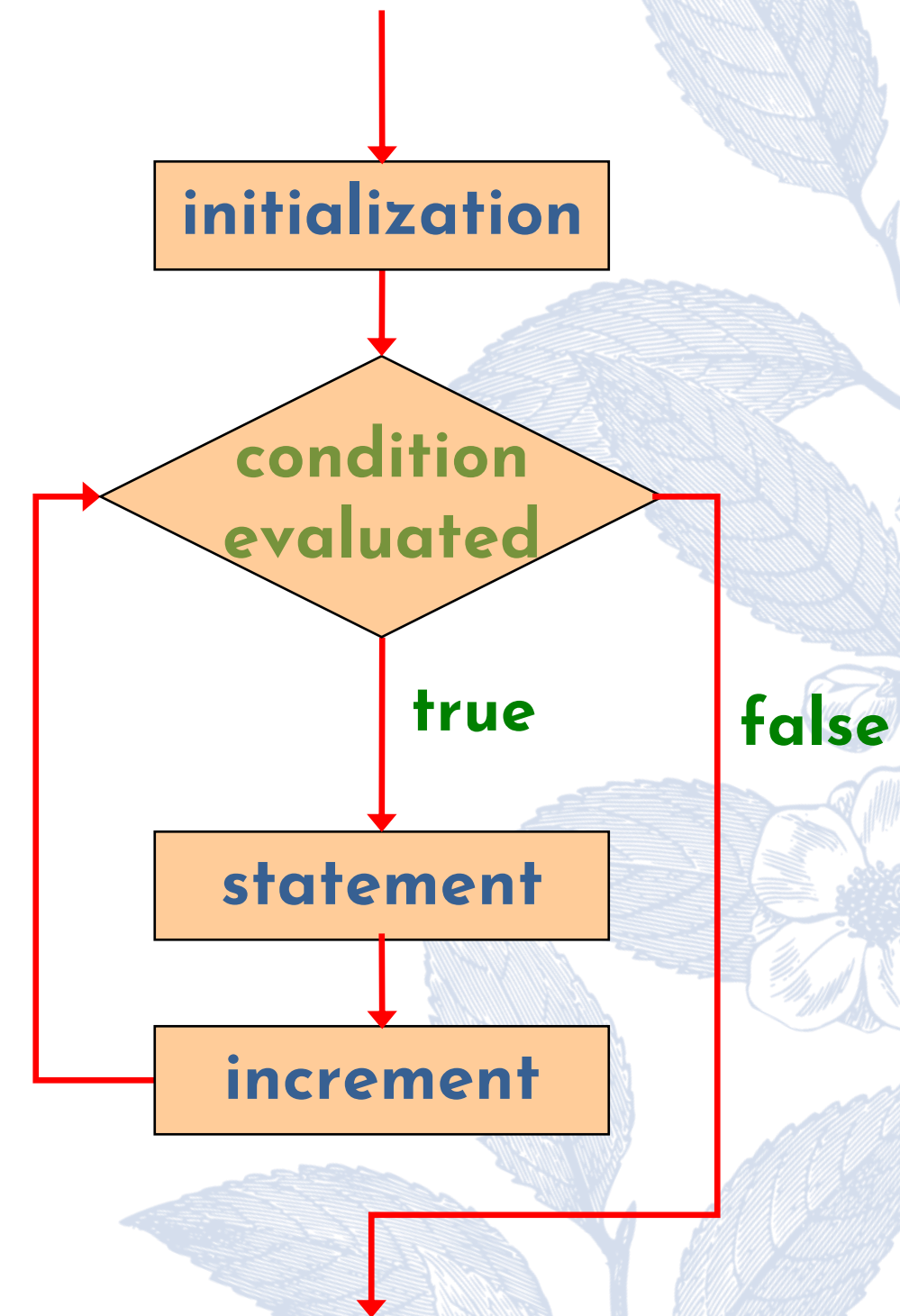✓A for statement has the following syntax:

The initialization is executed once before the loop begins

The statement is executed until the condition becomes false

for (**initialization** ; **condition** ; **increment** **)**
    statement;

The increment portion is executed at the end of each iteration

initialization

condition evaluated

true   false

statement

increment

The for Statement

✓A for loop is functionally equivalent to the following while loop
structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

For-each Loops

✓A variant of the for loop simplifies the repetitive processing of items in an iterator

✓For example, suppose bookList is an Array of Book object

✓The following loop will print each book:

```
for (Book myBook : bookList)
    System.out.println (myBook);
```

✓This version of a for loop is often called a for-each loop

✓A for-each loop can be used on any object that implements the Iterable interface

✓It eliminates the need to retrieve an iterator and call the hasNext and next methods explicitly

Iterators

✓An iterator is an object that allows you to process a collection of items one at a time

✓It lets you step through each item in turn and process it as needed

✓An iterator has a hasNext method that returns true if there is at least one more item to process

✓The next method returns the next item

✓Several classes in the Java standard class library are iterators

✓Ex: The Scanner class is an iterator

Chapter 1 - Java Fundamentals

Chapter Goals

✓To become familiar with your computing environment and your compiler

✓To compile and run your first Java program

✓To recognize syntax and logic errors

✓To write pseudocode for simple algorithms

✓To learn about Data and Expressions

✓To learn about Predefined classes

✓To learn about Conditionals and Loops

✓To learn about Arrays
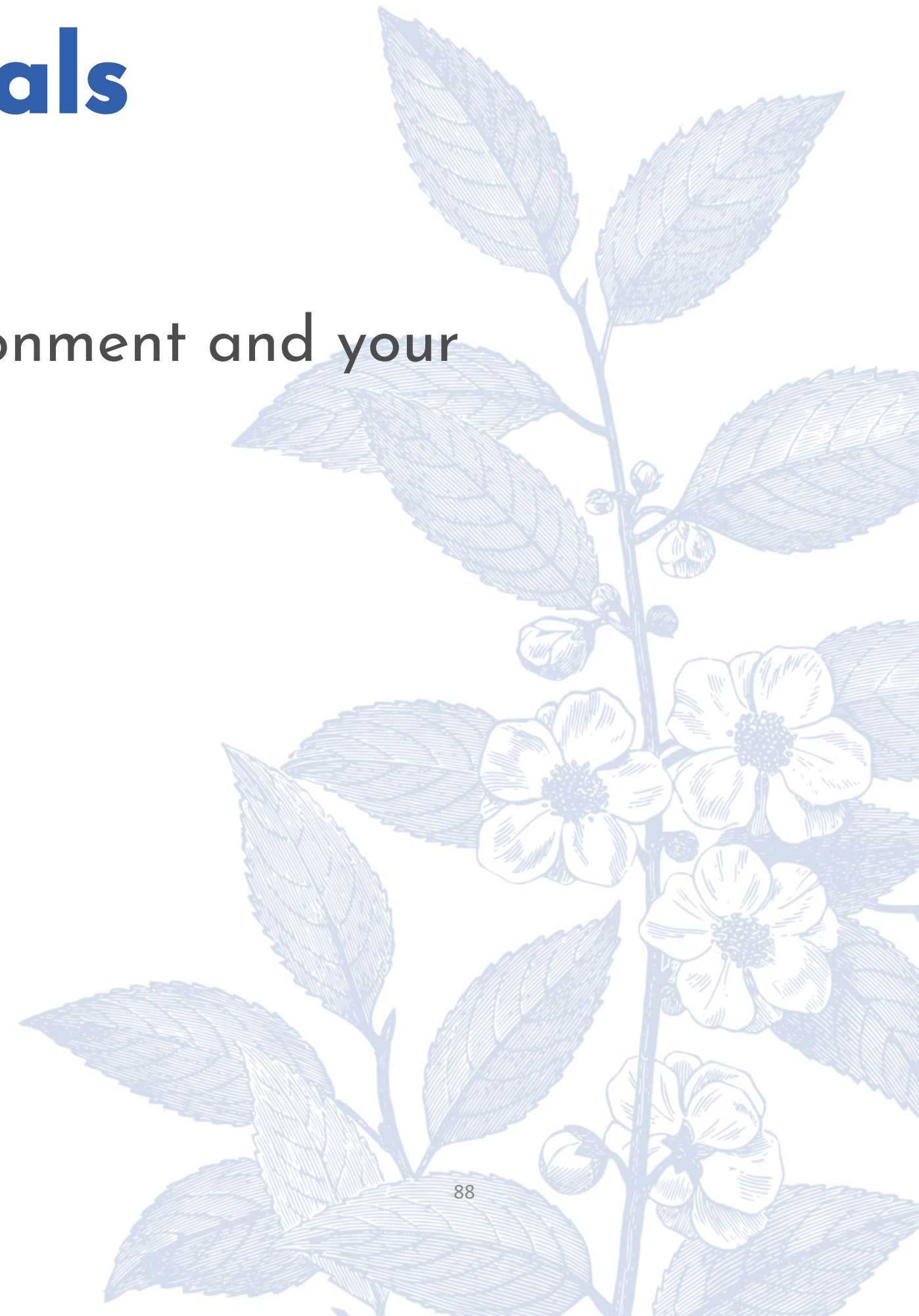
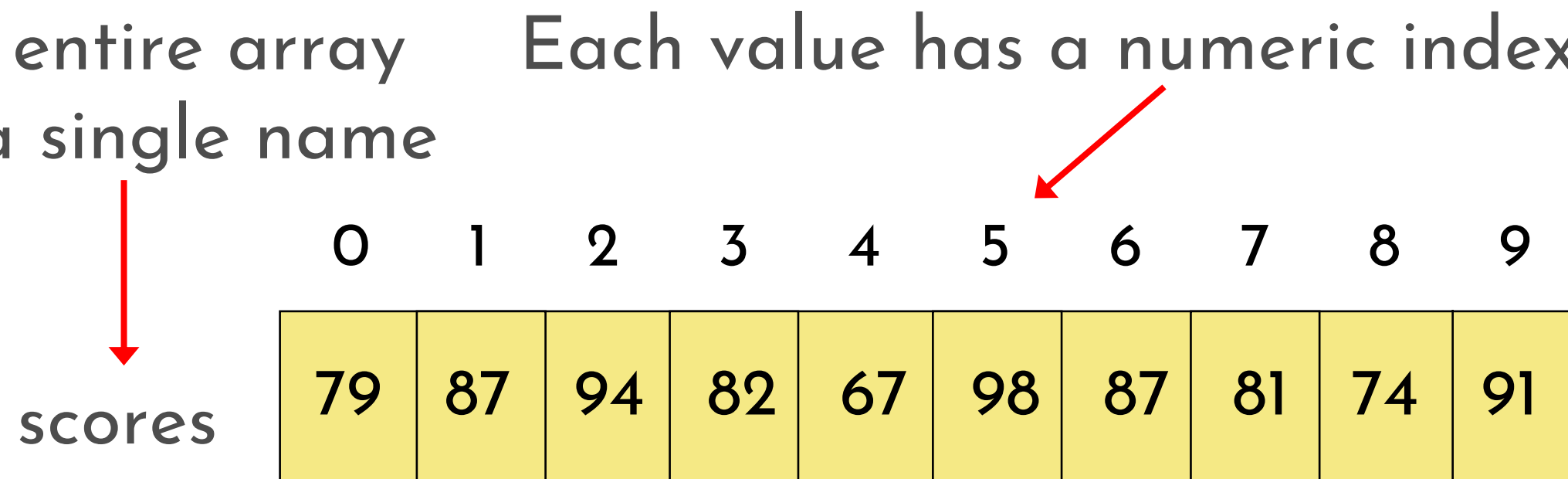# Arrays

✓An array is an ordered list of values:

The entire array has a single name

Each value has a numeric index

0	1	2	3	4	5	6	7	8	9
79	87	94	82	67	98	87	81	74	91

scores

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

✓A particular value in an array is referenced using the array
   name followed by the index in brackets

✓For example, the expression:
       scores[2]
       refers to the value 94 (the 3rd value in the array)

✓The values held in an array are called array elements

✓An array stores multiple values of the same type – the element
   type

✓The element type can be a primitive type or an object
   reference

Declaring Arrays

✓The scores array could be declared as follows:

   int[] scores = new int[10];

✓The type of the variable scores is int[] (an array of integers)

✓Note that the array type does not specify its size, but each object of that type has a specific size
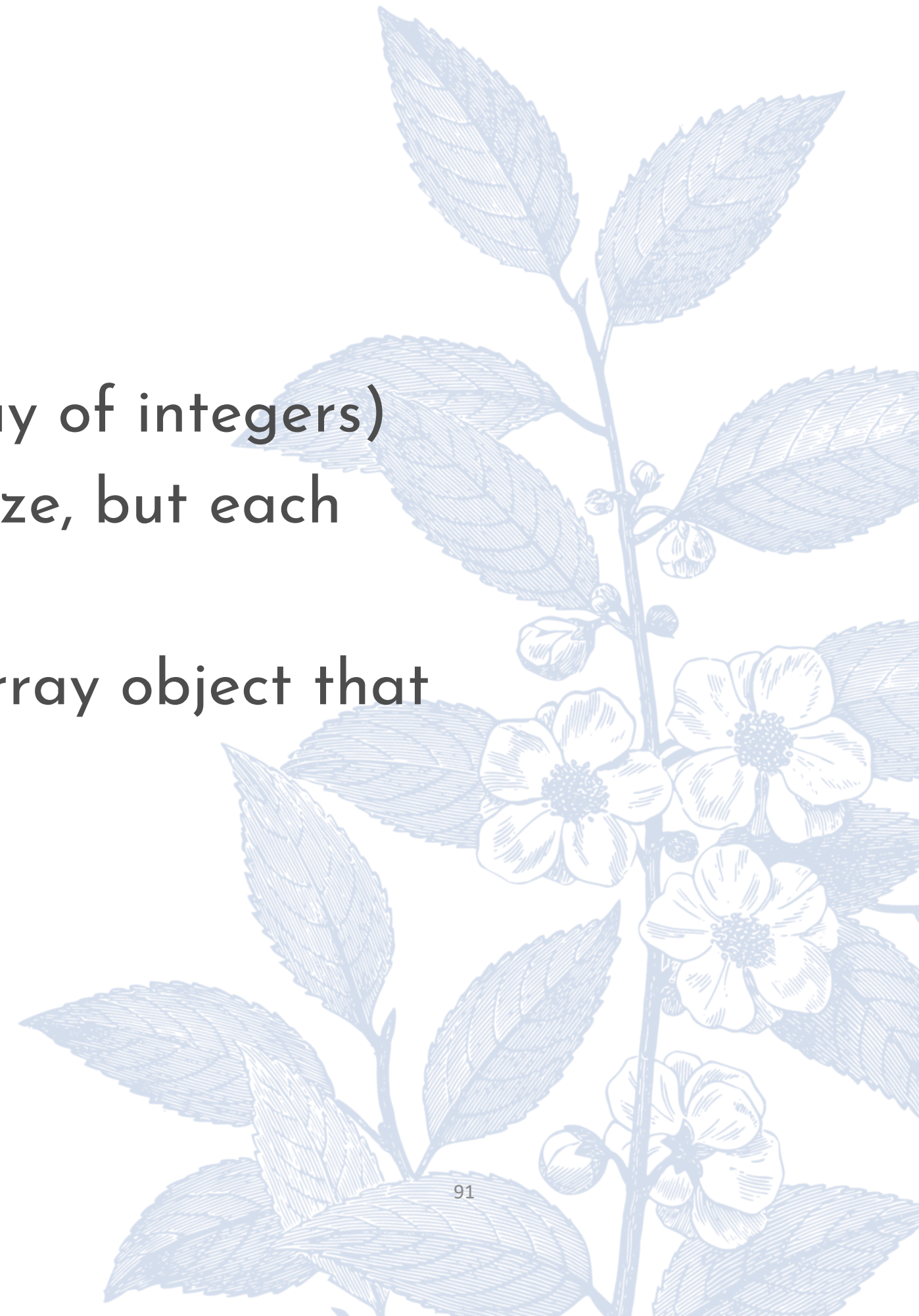
✓The reference variable scores is set to a new array object that can hold 10 integers

✓Ex:

   double[] prices = new double[500];
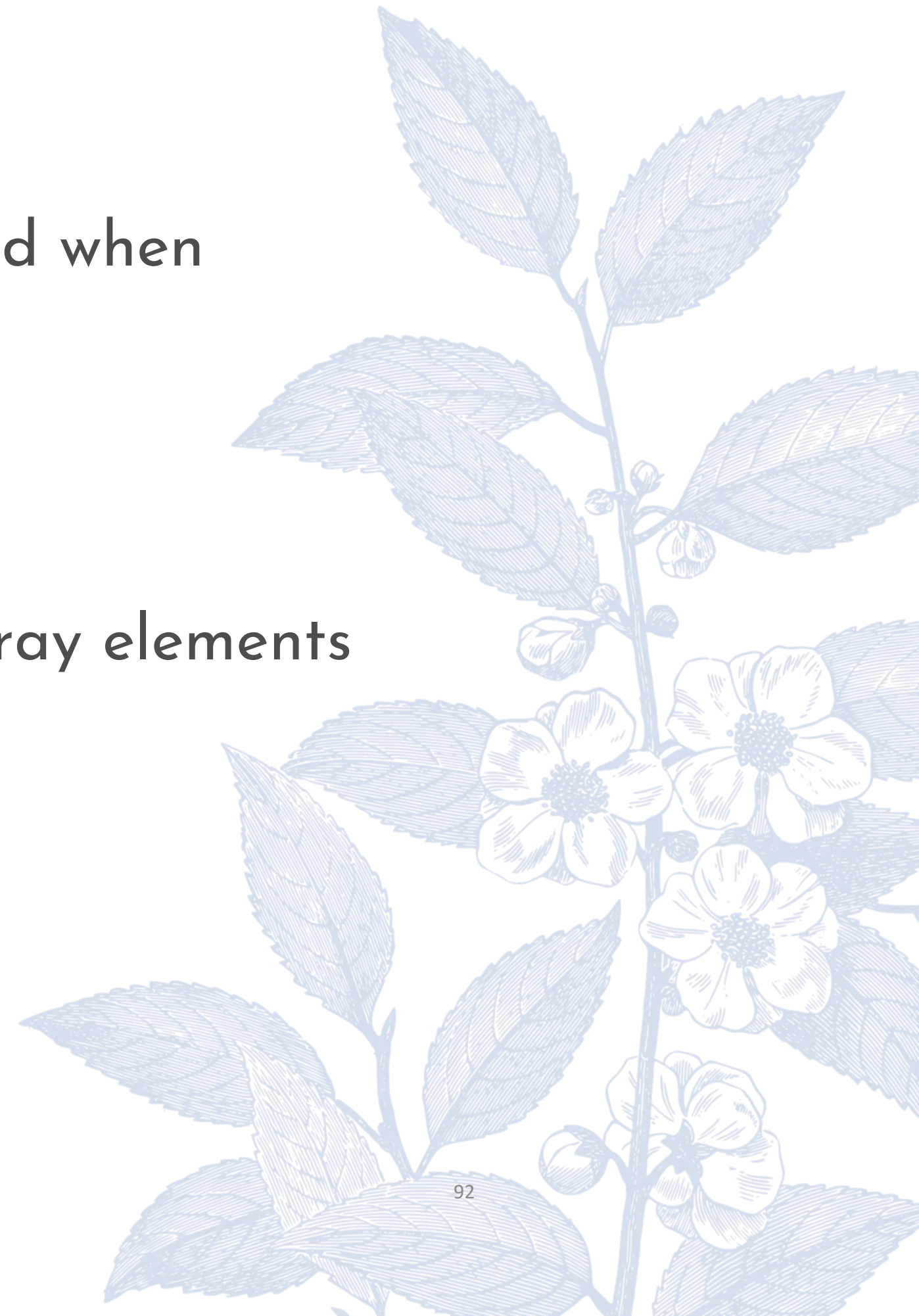
   boolean[] flags;

   flags = new boolean[20];

Using Arrays

✓The for-each version of the for loop can be used when processing array elements:

```
for (int score : scores)
    System.out.println (score);
```

✓This is only appropriate when processing all array elements starting at index 0

✓It can't be used to set the array values

Bounds Checking

✓Once an array is created, it has a fixed size

✓An index used in an array reference must specify a valid element

✓That is, the index value must be in range 0 to N-1

✓The Java interpreter throws an ArrayIndexOutOfBoundsException if an array index is out of bounds

✓This is called automatic bounds checking

✓Each array object has a public constant called length that stores the size of the array

Alternate Array Syntax

✓The brackets of the array type can be associated with the element type or with the name of the array

✓Therefore the following two declarations are equivalent:

double[] prices;

double prices[];

✓The first format generally is more readable and should be used

Initializer Lists

✓An initializer list can be used to instantiate and fill an array in one step

✓The values are delimited by braces and separated by commas

✓Examples:

   int[] units = {147, 323, 89, 933, 540, 269, 97, 114, 298, 476};

   char[] grades = {'A', 'B', 'C', 'D', 'F'};

✓Note that when an initializer list is used:

  ▪the new operator is not used

  ▪no size value is specified

✓The size of the array is determined by the number of items in the list

✓An initializer list can be used only in the array declaration

Arrays of Objects

✓The elements of an array can be object references

✓The following declaration reserves space to store 5 references
 to String objects
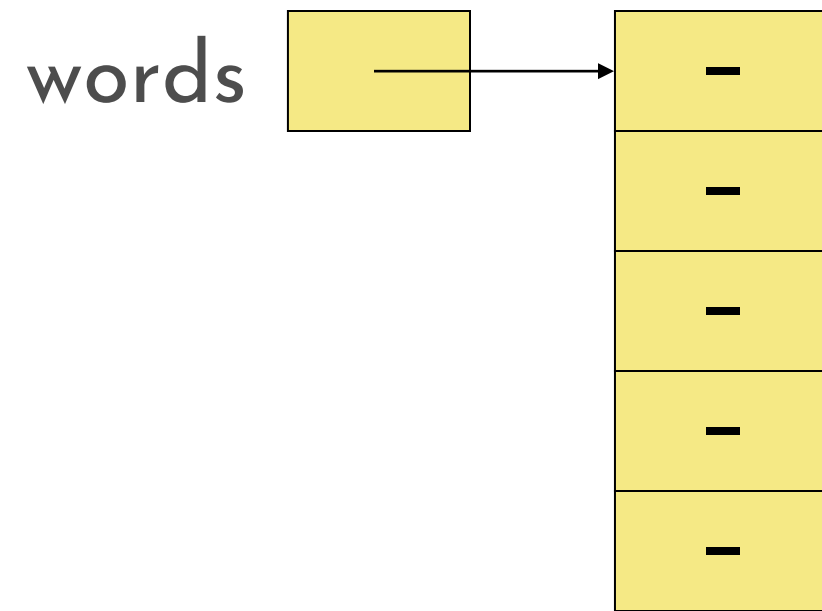
   String[] words = new String[5];

✓It does NOT create the String objects themselves

✓Initially an array of objects holds null references

✓Each object stored in an array must be instantiated separately

Arrays of Objects

✓The words array when initially declared:

words
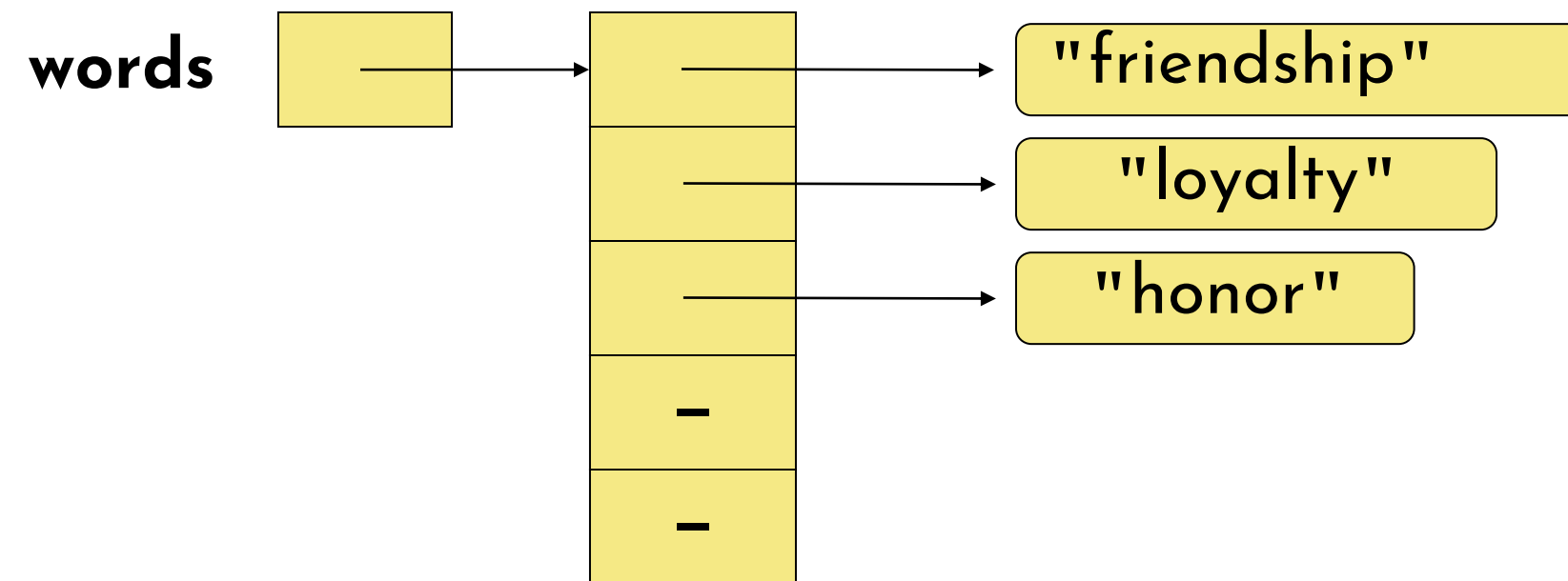
At this point, the following line of code would throw a

NullPointerException:

System.out.println(words[0]);

Arrays of Objects

✓After some String objects are created and stored in the array:
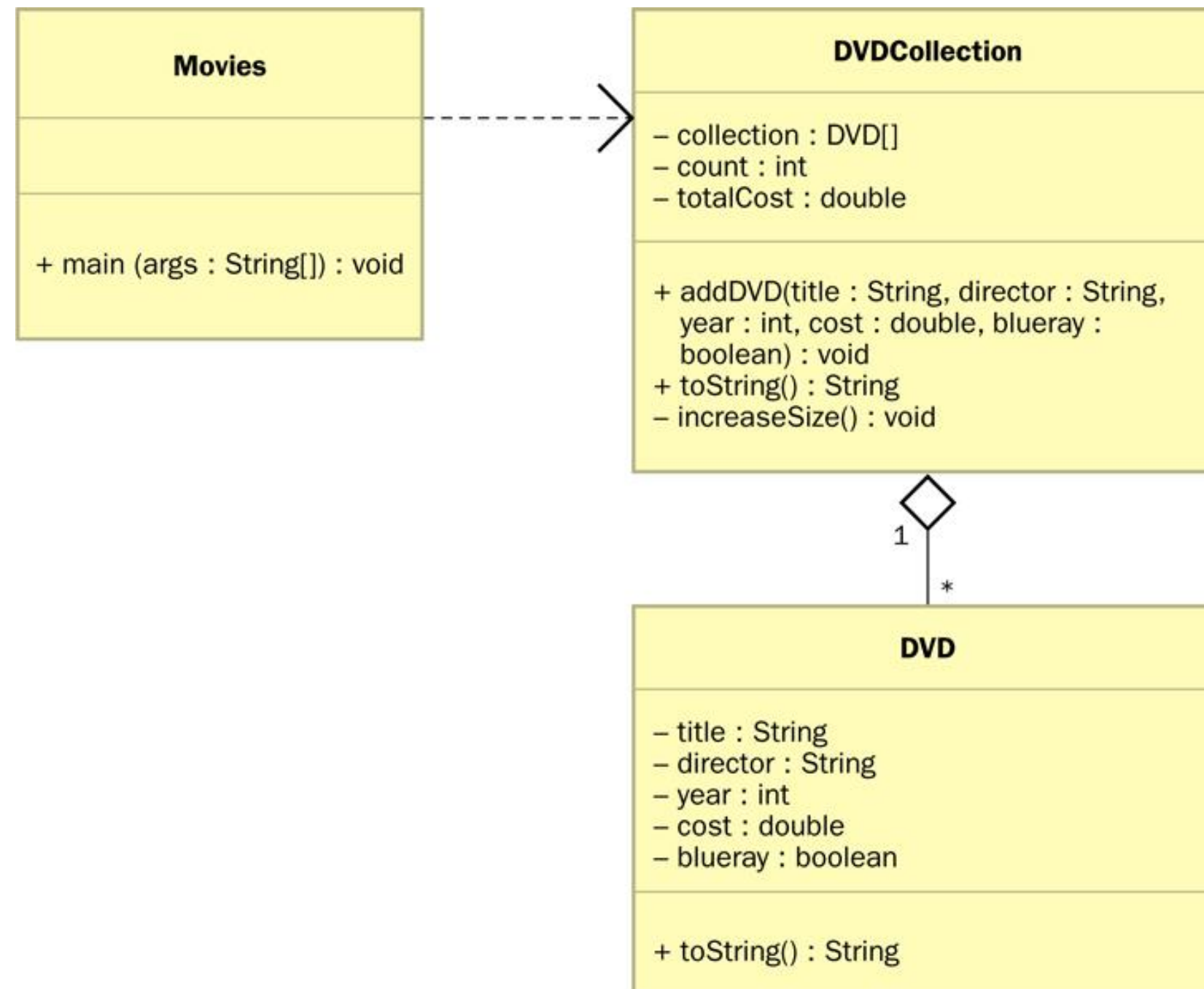
words

"friendship"

"loyalty"

"honor"

✓Keep in mind that String objects can be created using literals
String[] verbs = {"play", "work", "eat", "sleep", "run"};

Arrays of Objects

✓A UML diagram for the Movies program:

Movies

+ main (args : String[]) : void

DVDCollection

– collection : DVD[]
– count : int
– totalCost : double

+ addDVD(title : String, director : String,
   year : int, cost : double, blueray :
   boolean) : void
+ toString() : String
– increaseSize() : void

1

*

DVD

– title : String
– director : String
– year : int
– cost : double
– blueray : boolean

+ toString() : String

Command-Line Arguments

✓The signature of the main method indicates that it takes an array of String objects as a parameter

✓These values come from command-line arguments that are provided when the interpreter is invoked

✓For example, the following invocation of the interpreter passes three String objects into the main method of the StateEval program:

 java StateEval pennsylvania texas arizona

Variable Length Parameter Lists

✓Suppose we wanted to create a method that processed a different amount of data from one invocation to the next

✓For example, let's define a method called average that returns the average of a set of integer parameters

- one call to average three values

mean1 = average (42, 69, 37);

- another call to average seven values

mean2 = average (35, 43, 93, 23, 40, 21, 75);

Variable Length Parameter Lists

✓We could define overloaded versions of the average method

- Downside: we'd need a separate version of the method for each additional parameter

✓We could define the method to accept an array of integers

- Downside: we'd have to create the array and store the integers prior to calling the method each time

✓Instead, Java provides a convenient way to create variable length parameter lists

```
public double average (int … list)
{
   // whatever
}
```

Indicates a variable length parameter list

Two-Dimensional Arrays

✓A one-dimensional array stores a list of elements

✓A two-dimensional array can be thought of as a table of elements, with rows and columns

✓To be precise, in Java a two-dimensional array is an array of arrays

✓A two-dimensional array is declared by specifying the size of each dimension separately:

  int[][] table = new int[12][50];

✓A array element is referenced using two index values:

   value = table[3][6]

✓The array stored in one row can be specified using one index

Multidimensional Arrays

✓An array can have many dimensions – if it has more than one dimension, it is called a multidimensional array

✓Each dimension subdivides the previous one into the specified number of elements

✓Each dimension has its own length constant

✓Because each dimension is an array of array references, the arrays within one dimension can be of different lengths

✓these are sometimes called ragged arrays