Fundamentals of Java

A Simple Application

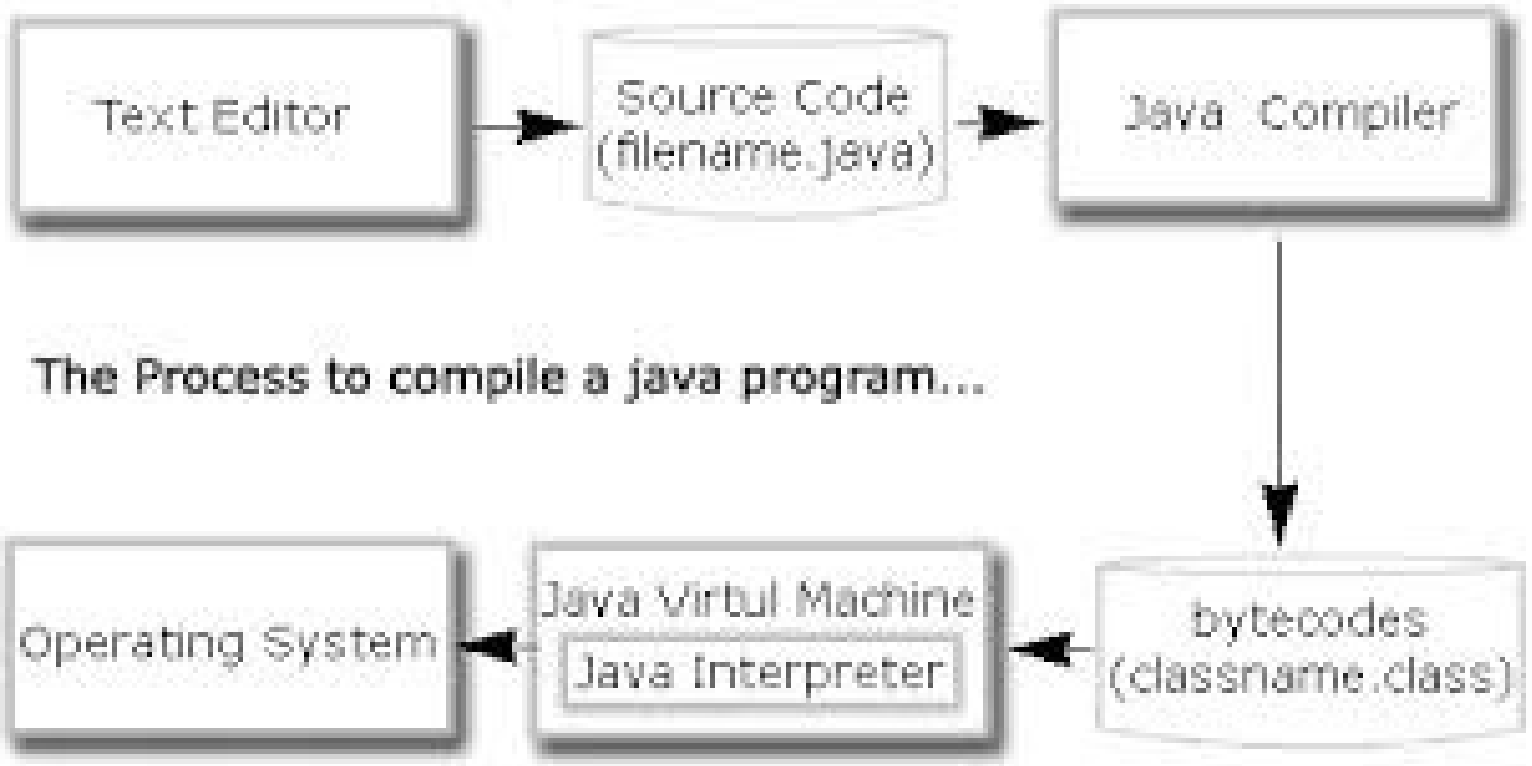
Example 1.1

```
//This application program prints Welcome
//to Java!

package chapter1;

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

Creating and Compiling Programs



Anatomy of a Java Program

- Comments
- Package
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method

Comments

In Java, comments are preceded by two slashes (//) in a line, or enclosed between /* and */ in one or multiple lines. When the compiler sees //, it ignores all text after // in the same line. When it sees /*, it scans for the next */ and ignores any text between /* and */.

Package

The second line in the program (package chapter1;) specifies a package name, chapter1, for the class Welcome.

Compiles the source code in Welcome.java, generates Welcome.class, and stores Welcome.class in the chapter1 folder.

Reserved Words

Reserved words or *keywords* are words that have a specific meaning to the compiler and cannot be used for other purposes in the program.

For example, when the compiler sees the word class, it understands that the word after class is the name for the class. Other reserved words in Example 1.1 are public, static, and void. Their use will be introduced later in the book.

Modifiers

Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used.

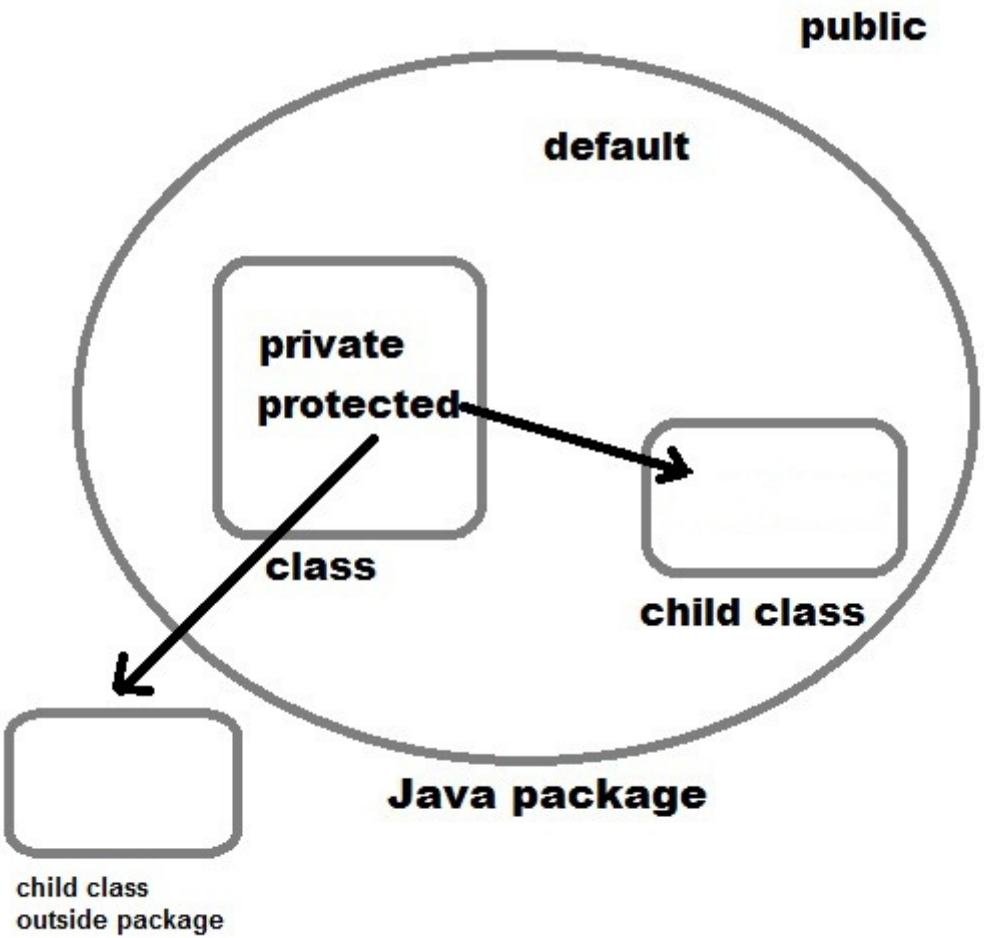
Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected.

A public datum, method, or class can be accessed by other programs.

A private datum or method cannot be accessed by other programs.

Access Control Modifiers

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).


public
default
private
protected
class
child class
Java package
child class
outside package

Non Access Modifiers

- The *static* modifier for creating class methods and variables
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.
- The *synchronized* and *volatile* modifiers, which are used for threads.

Final

Final

Final modifier is used to declare a field as final i.e. it prevents its content from being modified. Final field must be initialized when it is declared.

Example :

```
class Cloth
{
 final int MAX_PRICE = 999;    //final variable
 final int MIN_PRICE = 699;
 final void display()      //final method
 {
  System.out.println("Maxprice is" + MAX_PRICE );
  System.out.println("Minprice is" + MIN_PRICE);
 }
}
```

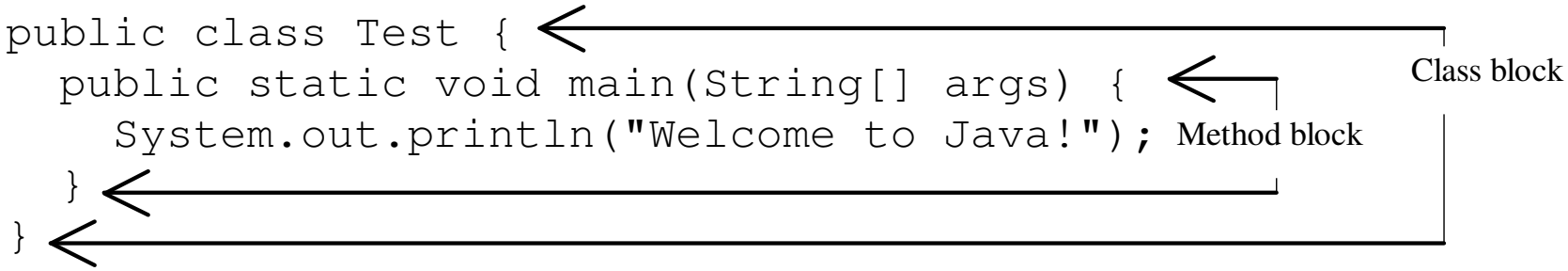
A class can also be declared as final. A class declared as final cannot be inherited. **String** class in java.lang package is a example of final class. Method declared as final can be inherited but you cannot override(redefine) it.

Statements

A *statement* represents an action or a sequence of actions. The statement System.out.println("Welcome to Java!") in the program in Example 1.1 is a statement to display the greeting "Welcome to Java!" Every statement in Java ends with a semicolon (;).

Blocks

A pair of braces in a program forms a block that groups components of a program.



Classes

The *class* is the essential Java construct. A class is a template or blueprint for objects. To program in Java, you must understand classes and be able to write and use them. The mystery of the class will continue to be unveiled throughout this book. For now, though, understand that a program is defined by using one or more classes.

Methods

What is System.out.println? It is a *method*: a collection of statements that performs a sequence of operations to display a message on the console. It can be used even without fully understanding the details of how it works. It is used by invoking a statement with a string argument. The string argument is enclosed within parentheses. In this case, the argument is "Welcome to Java!" You can call the same println method with a different argument to print a different message.

main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

```
public static void main(String[] args) {
  // Statements;
}
```

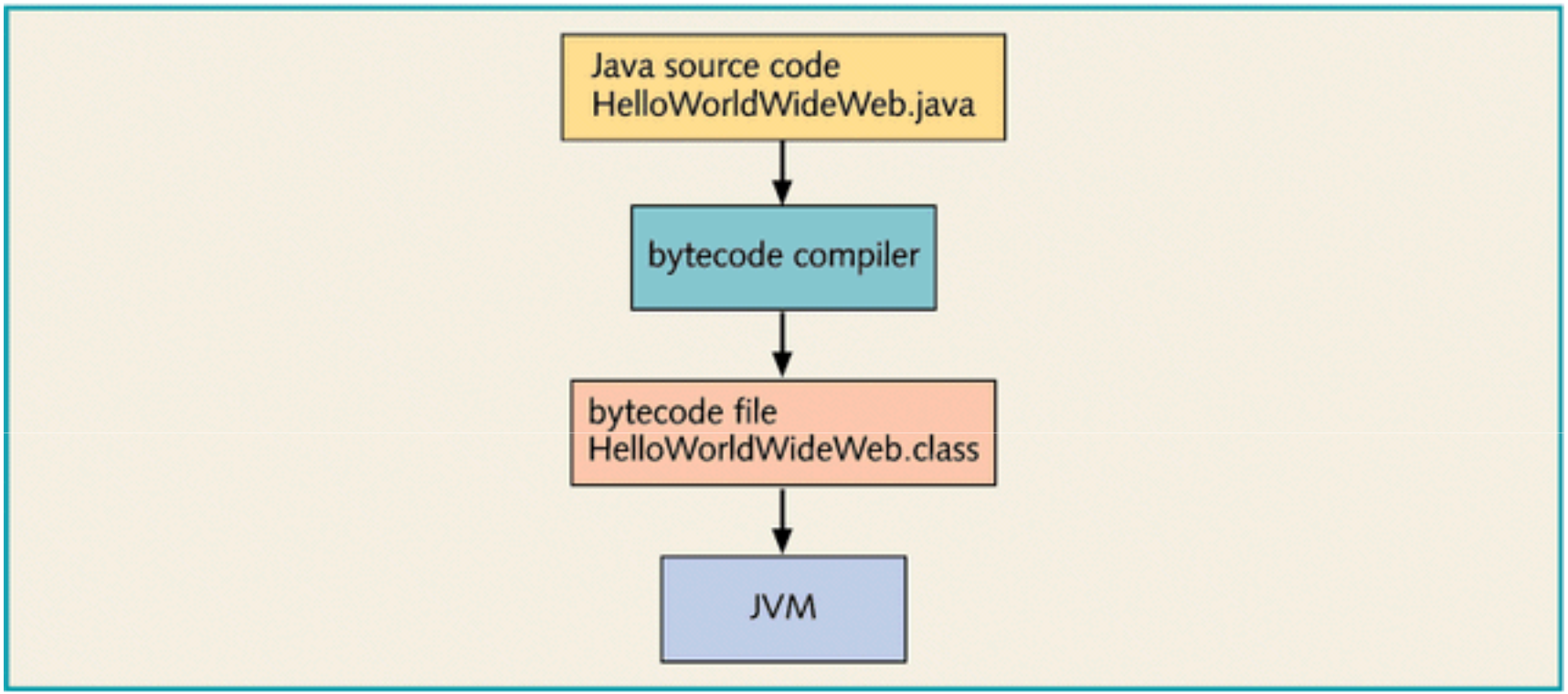


Figure 2-1 Compiling Java

Introducing Java

- Development environments
  - Java Development Kit
    - Available free from Sun Web site: java.sun.com
    - Includes: compiler JVM and prewritten classes
  - Integrated Development Environments (IDEs)
    - Provide:
      - Sophisticated editors
      - Debugging tools
      - Graphical development tools

Building a Java Class

- Each source code file defines a class
  - Class
    - HelloWorldWideWeb
  - File
    - HelloWorldWideWeb.java

Building a Java Class

- Class header
  - Describes class contained in source code file
  - Keywords:
    - public
      - Indicates class has public availability
    - class
      - Indicates line of code is a class header

Building a Java Class

- Identifiers
  - Name of a class, method, or variable
    - Can be any length
    - Can include any character except a space
    - Must begin with a letter of the alphabet, a dollar sign ($), or the underscore (_) character
  - Java is case sensitive
    - *Public* isn't the same as *public*

Building a Java Class

- Block of code
  - Used to group statements
  - Delineated by open curly brace ({) and closed curly brace (})
  - All code in Java is enclosed in a single block of code, which can contain additional blocks

Building a Java Class

- Indentation
  - Not required $\rightarrow$ recommended
- Line continuation
  - Can extend statements over multiple lines
    - No continuation character required

Building a Java Class

- Comments
  - Single line
    - // compiler ignores everything to end of line
  - Multi-line
    - /* compiler ignores everything in between */
  - Multi-line (documentation)
    - /** compiler ignores everything in between */
    - Used for JavaDoc

Building a Java Class

- Java code generally consists of:
  - Variable definitions
  - One or more methods
- Method header
  - Comments to identify method and describe some of its characteristics

Building a Java Class

- Argument
  - Information sent to a method
  - Contained in parentheses of method call
- Literal
  - Value defined within a statement
- Semicolon (;)
  - All java statements end with a semicolon

Using Java Variables and Data Types

- Variable
  - Name of place in memory that can contain data
  - All variables have:
    - Data type → kind of data variable can contain
    - Name → identifier that refers to the variable
    - Value → the default or specified value

Using Java Variables and Data Types

- Declaring and Initializing Variables
  - Variable data type must be declared prior to initialization
    - Eight available primitive data types
  - Assignment operator (=)
    - Used to assign value to a variable
      - char c = ‘a’;
      - boolean b = true;
      - double d = 1.25;

Table 2-3 Java Primitive Data Types

	Type	Range of Values	Size
Numeric with no decimals	1. int	+ or – 2.1 trillion	4 bytes
	2. short	+ or – 32,000	2 bytes
	3. long	+ or – 9 E18	8 bytes
	4. byte	+ or – 127	1 byte
Numeric with decimals	5. double	+ or – 1.79 E308	8 bytes, 15 decimals
	6. float	+ or – 3.4 E38	4 bytes, 7 decimals
Other	7. boolean	true or false	
	8. char	any character	2 bytes

Using Java Variables and Data Types

- Changing Data Types
  - If changing data type results in no loss of precision, can be done implicitly:

```
int c = 5; double a, b = 3.5;
a = b + c;
```

  - *Casting* allows data type changes explicitly with loss of precision:

```
int a, c = 5; double b = 3.5;
a = (int) b + c;
```

Using Java Variables and Data Types

- Using Constants
  - Variable with a value that doesn't change
  - Keyword
    - final
      - Denotes value cannot change
  - Example:
    - final double SALES_TAX_RATE = 4.5;

Using Java Variables and Data Types

- Using Reference Variables
  - Java has two types of variables:
    - Primitive
    - Reference
      - Uses class name as a data type
      - Points to an instance of that class
      - Example:
        - » String s = “Hello World”;

Using Java Variables and Data Types

- Creating a Java Class to Demonstrate Variables
  - Invoking a method
    - Send the method a message asking it to execute
  - Concatenation operator (+)
    - Joins String literals and variables
    - Automatically converts numeric and Boolean values to strings before use in *println* method

Computing with Java

- Operators
  - Arithmetic operators (+,-,*,/)
    - addition, subtraction, multiplication, division
    - Precedence using parentheses
  - Remainder (modulus) operator (%)
    - Produces remainder from division of two integers
- Math Class
  - Methods for exponentiation, rounding, etc.

Table 2-4 Java Arithmetic Operators

Operator	Description	Example	Result
+	addition	11 + 2	13
–	subtraction	11 –2	9
*	multiplication	11 * 2	22
/	division	11 / 2	5
%	remainder	11 % 2	1

Table 2-5 Selected Math Class Methods

Method	Description
`abs(x)`	Returns the absolute value of `x`
`max(x,y)`	Returns the greater of `x,y`
`min(x,y)`	Returns the smaller of `x,y`
`pow(x,y)`	Returns the value of `x` raised to the power of `y`
`random()`	Returns a random number between 0 and 1
`round(x)`	Returns the closest integer value to x
`sqrt(x)`	Returns the square root of `x`

Computing with Java

- Special Operators
  - For writing shortcut code
    - Increment operator (++)
      - Add one to a variable
    - Decrement operator (--)
      - Subtract one from a variable
    - Assignment operator with arithmetic operators:

      total = total + 5;

      Total += 5;

Writing Decision-Making Statements

- Decision Making Statement
  - Determine whether a condition is true, and take some action based on determination

- Three ways to write decision-making statements:
  - if statement
  - switch statement
  - conditional operator

Writing Decision-Making Statements

- Writing *if* Statements
  - *if* statement:
    - Interrogates logical expression enclosed in parentheses
    - Determines whether it is true or false
    - Uses logical operators to compare values:
      - e.g., (studentAge < 21)

Table 2-6 Java Logical Operators

Operator	Description
&&	And
==	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
!	Not
!=	not equal to
||	Or

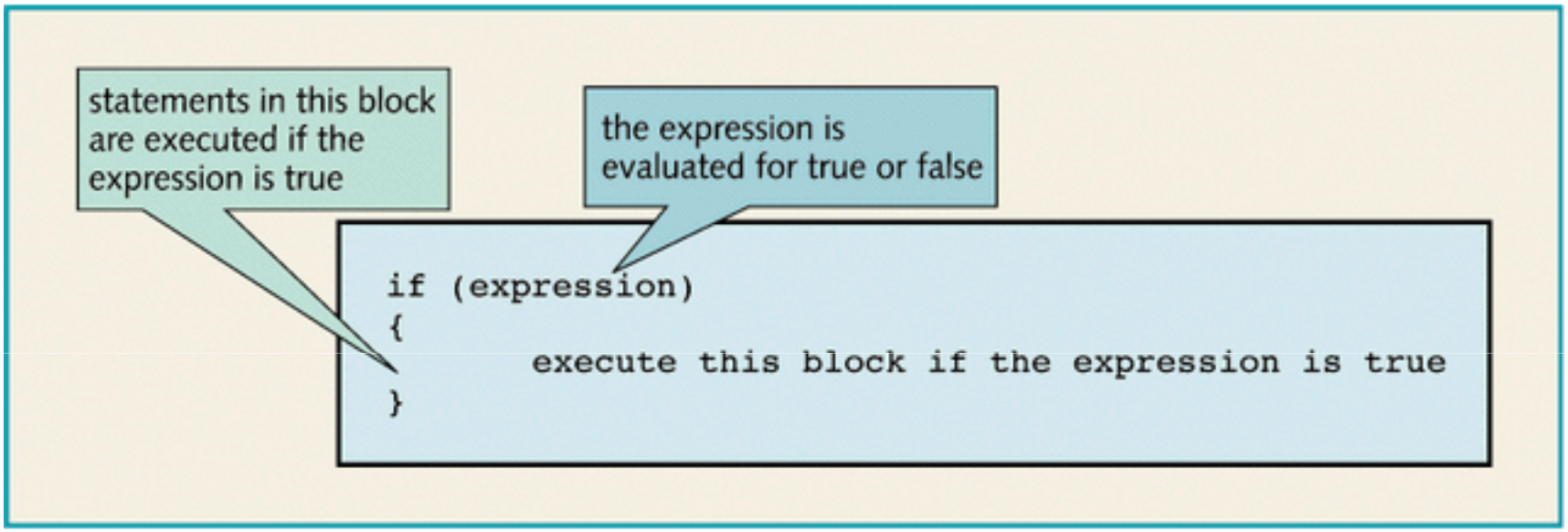


Figure 2-12 if statement format

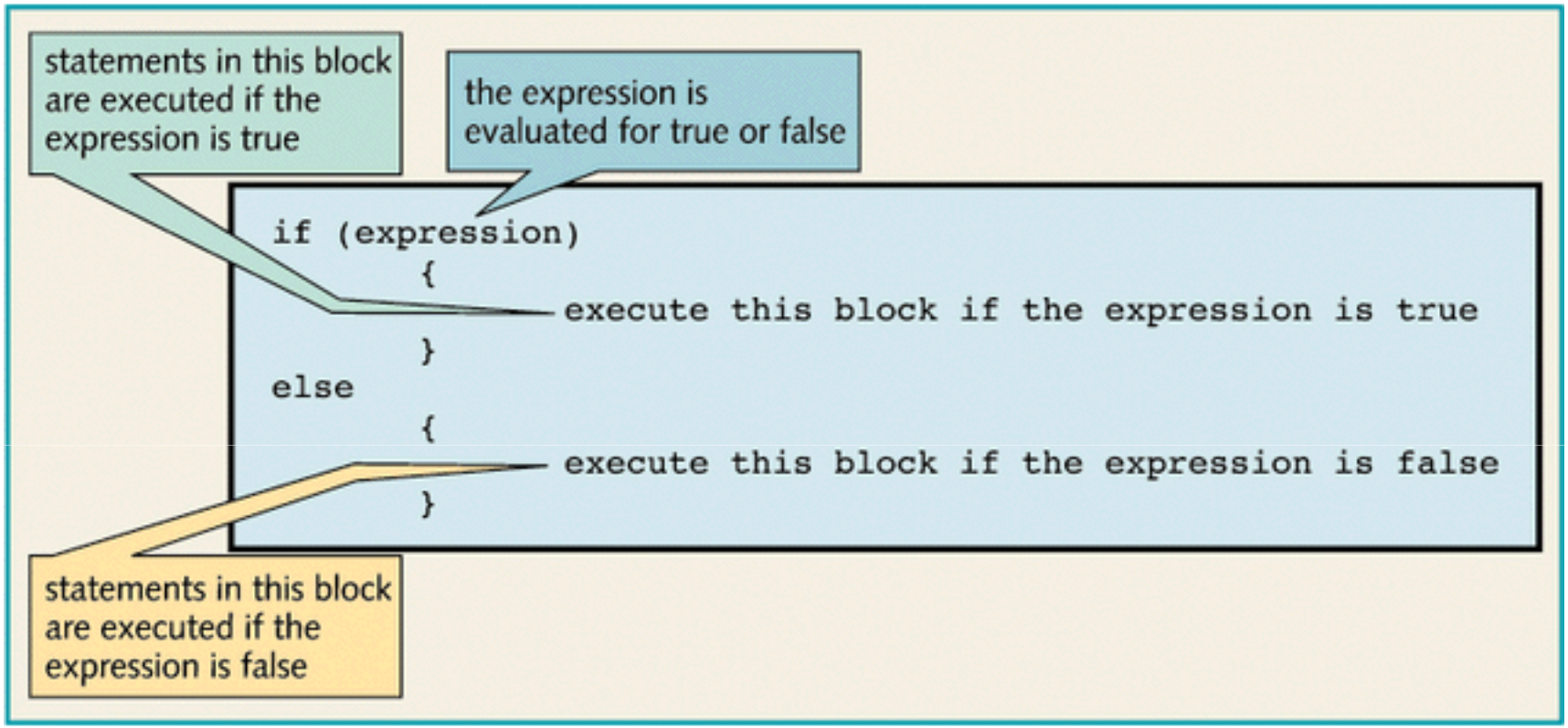


Figure 2-13 if-else statement format

Writing Decision-Making Statements

- Writing *if* Statements
  - Compound expression
    - Two expressions joined using logical operators
      - OR → ||
      - AND → &&
  - Nested *if* statement
    - *if* statement written inside another *if* statement

Writing Decision-Making Statements

- Using the Conditional Operator
  - Conditional operator (?)
    - Provides a shortcut to writing an if-else statement
    - Structure:
      - variable = expression ? value1:value2;

Writing Decision-Making Statements

- Writing *switch* Statements
  - Acts like a multiple-way *if* statement
  - Transfers control to one of several statements or blocks depending on the value of a variable
  - Used when there are more than two values to evaluate
  - Restrictions:
    - Each case evaluates a single variable for equality only
    - Variable being evaluated must be: char, byte, short, or int

Writing Loops

- Loops
  - Provides for repeated execution of one or more statements until a terminating condition is reached
  - Three basic types:
    - while
    - do
    - for

Writing Loops

- Writing *while* Loops
  - Loop counter
    - Counts number of times the loop is executed
  - Two kinds of loops
    - Pre-test loop
      - Tests terminating condition at the beginning of the loop
    - Post-test loop
      - Tests terminating condition at the end of the loop

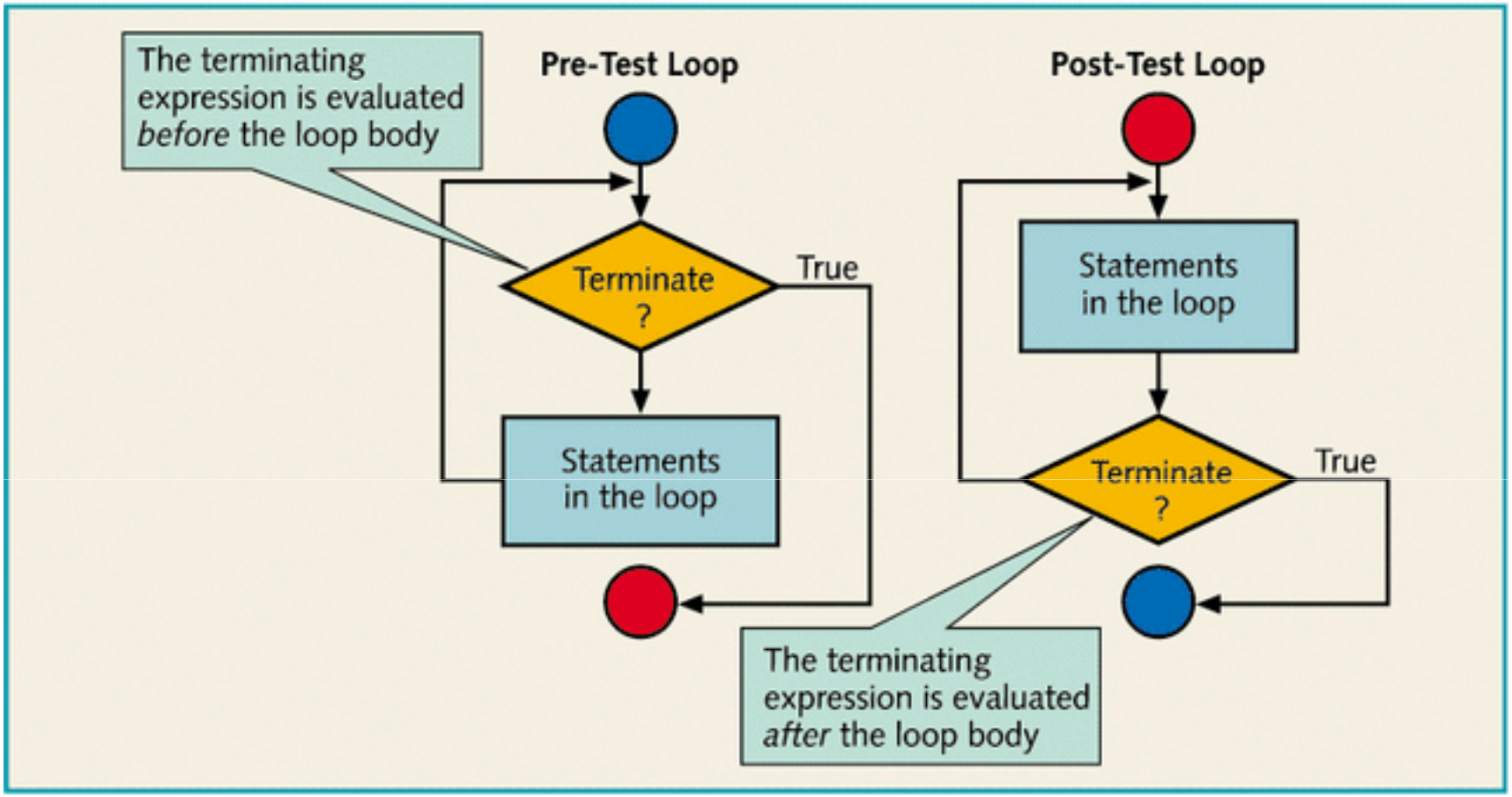


Figure 2-16 Loop structures

Writing Loops

- Writing *do* Loops
  - Loop counter
    - Counts number of times the loop is executed
  - Post-test loop
    - Tests terminating condition at the end of the loop
    - Forces execution of statements in the loop body at least once

Writing Loops

- Writing *for* Loops
  - Loop counter
    - Counts number of times the loop is executed
  - Pre-test loop
    - Tests terminating condition at the beginning of the loop
  - Includes counter initialization and incrementing code in the statement itself

Writing Loops

- Writing Nested Loops
  - A loop within a loop
  - Useful for processing data arranged in rows and columns
  - Can be constructed using any combinations of *while*, *do*, and *for* loops

Declaring and Accessing Arrays

- Arrays
  - Allows the creation of a group of variables with the same data type
  - Consist of elements:
    - Each element behaves like a variable
  - Can be:
    - One dimensional
    - Multi-dimensional

Declaring and Accessing Arrays

- Using One-Dimensional Arrays
  - Keyword
    - new
      - Used to create a new array instance
      - int testScores[] = new int[10];
  - Use brackets ([]) and indices to denote elements:
    - testScores[5] = 75;

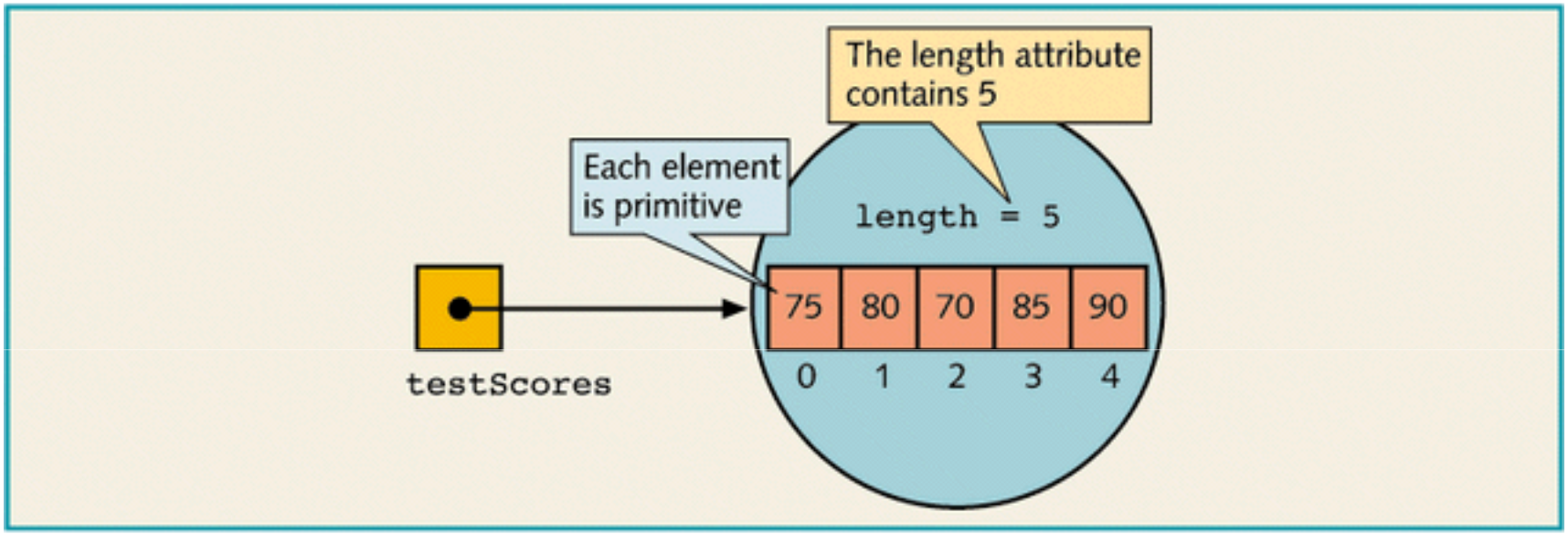


Figure 2-21 A five-element `int` array

Declaring and Accessing Arrays

- Using Multidimensional Arrays
  - Array of arrays
    - Three dimensions → cube
    - Four dimensions → ???
  - Each dimension has its own set of brackets:
    - testScoreTable[5][5] = 75;

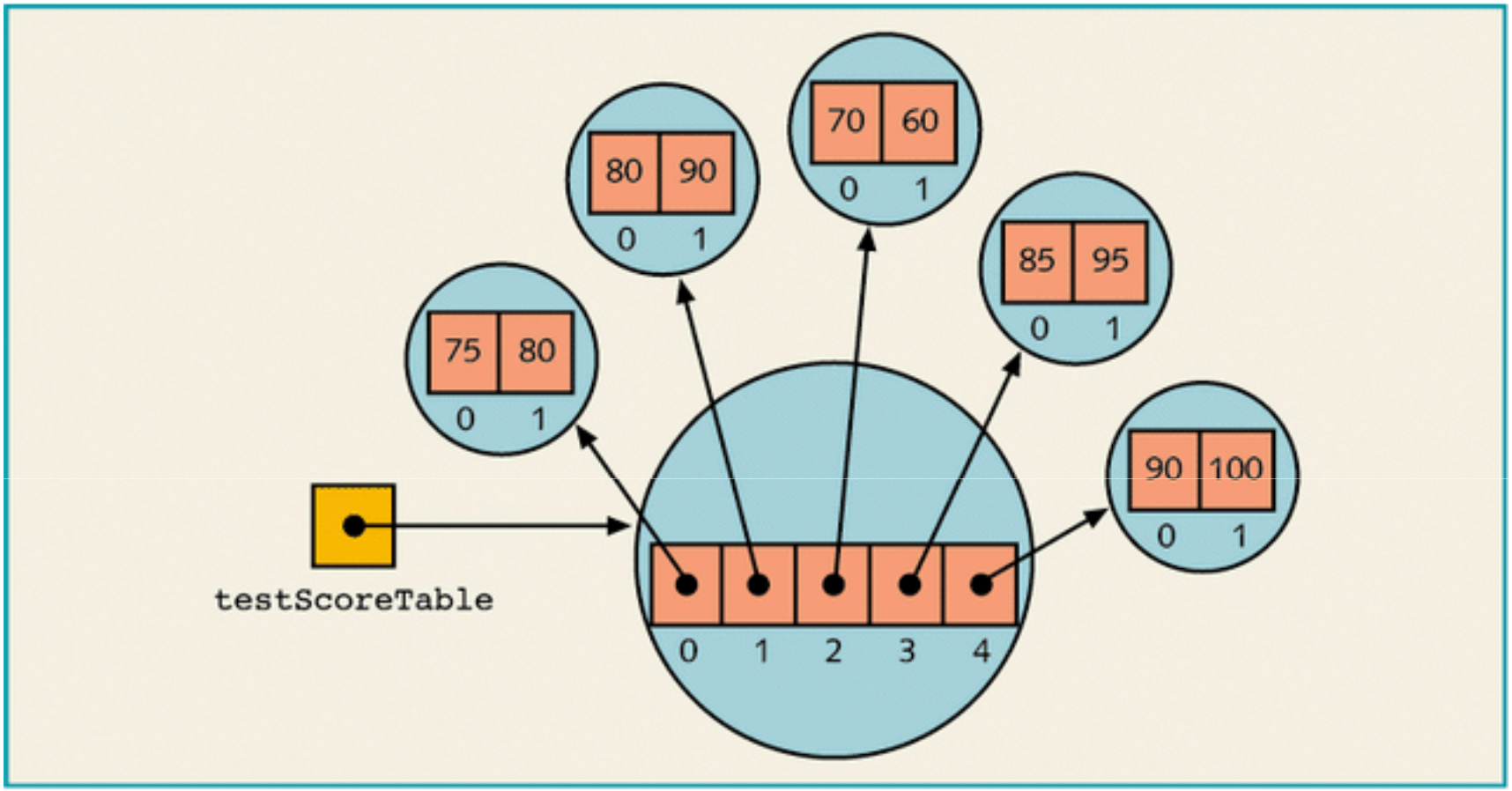


Figure 2-24 An array of arrays

Building a Java Clss

- Previous lab and homework demonstrated a simple Java application.

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

- Be familiar with the basic components of this sample. All applications will have this framework.
  Remember: Class names should begin with a capital.

Building a Java Class

- Java code generally consists of:
  - Comments
  - Variable definitions
    - e.g.
      - int aNumber = 1;
      - boolean isFound = true;
      - double anotherNumber = 1.50;
      - String s = “example of a string”;
    - note the naming conventions
  - Method definitions
    - e.g.
      - the main method in applications
        public static void main (String[] args) {
        }
      - public computeTax();
    - again note the naming conventions

Building a Java Class

- Comments
  - Single line
    - // compiler ignores everything to end of line
  - Multi-line
    - /* compiler ignores everything in between */
  - Multi-line (documentation)
    - /** compiler ignores everything in between */
    - Used for JavaDoc
  - Comments do not appear in executable code

Variable Definitions

- Variable definitions
  - Variable: name of place in memory that can contain data
  - All variables have:
    - Data type $\rightarrow$ kind of data variable can contain
    - Name $\rightarrow$ identifier that refers to the variable
    - Value $\rightarrow$ the default or specified value
      - also called the literal of the statement
    - Semicolon
      - Remember: All java statements end with a semicolon!
  - e.g.
    - String s = “MC697”;
    - int count = 5;

Variable Definitions

- Initializing Variables
  - Assignment operator (=)
    - Used to assign value to a variable
      - char c = ‘a’; - note the single quotes
      - boolean b = true;
      - double d = 1.25;
    - Important: This is different from the comparative equals (==)
    - If variable is not initialized, most will default to null.
      All variables should be initialized.

Data Types

- Declaring Variables
  - Variable data type must be declared prior to initialization
  - There are two basic data types:
    - Primitive data types
      - Eight available primitive data types
      - Primitive data types are not capitalized in variable declarations
        - » int aNumber = 5;
    - Reference data types
      - These data types are capitalized.
        - » String s = “example string”;

Table 2-3 Java Primitive Data Types

	Type	Range of Values	Size
Numeric with no decimals	1. int	+ or – 2.1 trillion	4 bytes
	2. short	+ or – 32,000	2 bytes
	3. long	+ or – 9 E18	8 bytes
	4. byte	+ or – 127	1 byte
Numeric with decimals	5. double	+ or – 1.79 E308	8 bytes, 15 decimals
	6. float	+ or – 3.4 E38	4 bytes, 7 decimals
Other	7. boolean	true or false	
	8. char	any character	2 bytes

Data Types

- Using Reference Variables
  - Uses class name as a data type
  - Points to an instance of that class
  - Example:
    - String s = “Hello World”;
    - Employee emp1;

Variable Constants

- Using Constants
  - Variable with a value that doesn't change
  - Keyword
    - final
      - Denotes value cannot change
  - Example:
    - final double SALES_TAX_RATE = 4.5;
    - note the naming convention
    - other examples?

Method Defnitions

- Methods contain:
  - access modifier
    - defines who can call this method
  - return type
    - defines what this method will return
  - static
    - optional - we will discuss this later
  - method name
    - should begin with a lower case
  - argument definition section
    - defines what input is expected in this method
  - block of code with statements

Statements Within a Method

- This is the logic of the program.
  - Can define and assign variables.
  - Can invoke another method
    - Send the method a message asking it to execute
    - Message is sent in the argument section.
      - e.g. System.out.println(“output from method”);
      - What is the argument in this method?

Computing with Java

- Changing Data Types
  - If changing data type results in no loss of precision, can be done implicitly:

    ```
    int c = 5; double a, b = 3.5;
    a = b + c;
    ```

  - *What is it called when you explicitly change the data type?*

Computing with Java

– *Casting* allows data type changes explicitly with loss of precision:

```
int a, c = 5; double b = 3.5;
a = (int) b + c;
```

Computing with Java

- Operators
  - Arithmetic operators (+,-,*,/)
    - addition, subtraction, multiplication, division
    - Precedence using parentheses
    - Remainder (modulus) operator (%)
      - Produces remainder from division of two integers
    - Concatenation operator (+)
      - Joins String literals and variables
      - Automatically converts numeric and Boolean values to strings before use in *println* method
- Math Class
  - Methods for exponentiation, rounding, etc.

Table 2-4 Java Arithmetic Operators

Operator	Description	Example	Result
+	addition	11 + 2	13
–	subtraction	11 –2	9
*	multiplication	11 * 2	22
/	division	11 / 2	5
%	remainder	11 % 2	1

Table 2-5 Selected Math Class Methods

Method	Description
`abs(x)`	Returns the absolute value of `x`
`max(x,y)`	Returns the greater of `x,y`
`min(x,y)`	Returns the smaller of `x,y`
`pow(x,y)`	Returns the value of `x` raised to the power of `y`
`random()`	Returns a random number between 0 and 1
`round(x)`	Returns the closest integer value to x
`sqrt(x)`	Returns the square root of `x`

Computing with Java

- Special Operators
  - For writing shortcut code
    - Increment operator (++)
      - Add one to a variable
    - Decrement operator (--)
      - Subtract one from a variable
    - Assignment operator with arithmetic operators:

      total = total + 5;

      What is another way of writing this statement?