SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

3-hour lab

2-hour lab

3 x 1 hour lectures

1-hour Tutorial

Total of
9 Hours
Class time

Module Description

This module is intended to give the student a solid foundation in programming theory. It covers the theory of programming constructs and implements these in a lab-based environment for the student to fully grasp the theory and understand the practice.

Introduction to Programming

Learning Outcomes (Abbreviated)

On completion of this module the learner will/should be able to:

- Understand the basic concepts of the key programming structures
- Evaluate the most appropriate constructs to use and to implement these constructs in solving a variety of problems

Learning Outcomes (Abbreviated)

On completion of this module the learner will/should be able to:

- Understand a specification, prepare suitable data to test the specification
- Demonstrate competency in the fundamentals of developing software
- Work individually and as a member of a team

Module Assessment

Practical (Lab) Exams

#	Exam	Weight
1	Monday, Nov 11th – 2.5 hours, open book	15%
2	Monday, Dec 16th – 2.5 hours, open book	15%
3	Monday, Feb 24th – 2.5 hours, open book	15%
4	Monday, April 28th – 2.5 hours, open book	15%
5	Every Thursday, semester 2 (10 X 1%)	10%

Lab Exams Total: **70%**

Multiple Choice Quizzes

#	Quiz	Weight
1	Thursday, Nov 14th – 1 hour, closed book	7.5%
2	Thursday, Dec 12th – 1 hour, closed book	7.5%
3	Thursday, February 27th – 1 hour, closed book	7.5%
4	Friday, Apr 30th – 1 hour, closed book	7.5%

MCQ Total: **30%**

Module
Content

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

Program development is the process of conceiving, specifying, designing, programming, documenting, testing, and bug fixing involved in creating and maintaining software.

What is Programming?

A program is a sequence of instructions that specifies how to perform a computation on computer hardware. The computation might be something mathematical, like solving a system of equations or finding the roots of a polynomial.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

What is Programming?

A few basic instructions appear in just about every language:

input: Get data from the keyboard, a file, a sensor, or some other device.

output: Display data on the screen or send data to a file or other device.

math: Perform basic mathematical operations like addition and division.

decision: Check for certain conditions and execute the appropriate code.

repetition: Perform an action repeatedly, usually with some variation.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A brief history of Computer Programming

1995: Java

Java was developed by James Gosling and other developers at Sun Microsystems and was first introduced to the public in 1995. It is a general-purpose programming language.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Why Java?

Platform Independent
It is one of the biggest merits of java - Java is platform independent and can run on any platform such as Windows, Mac, Linux, etc. (WORA)

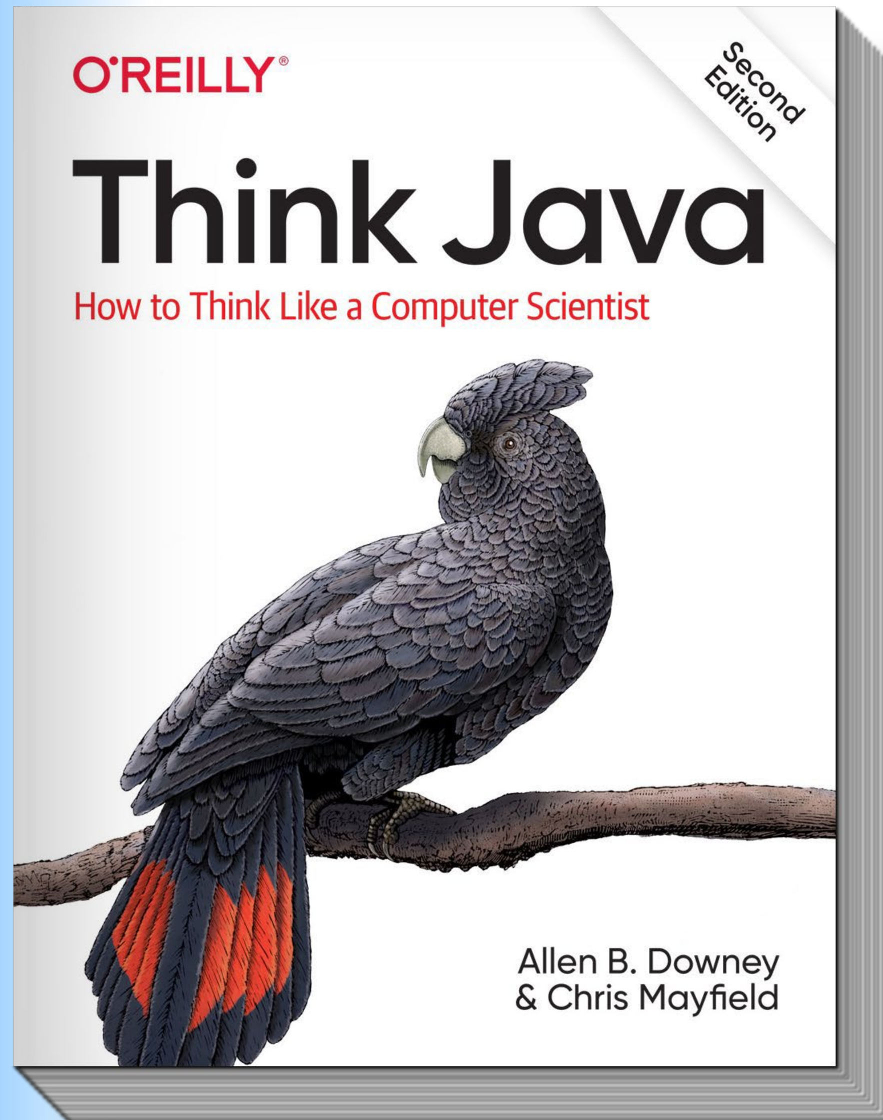
Secure
Java is considered to be a relatively secure language

Simple
Java is easier to learn compared to languages such as C or C++.

Object Oriented
Java is an *object-oriented* programming language and lends itself well to creating reusable code.

Robust
There are many features like automatic garbage collection, type checking and exception handling that makes java a robust (strong) language.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT



Think Java is a hands-on introduction to computer science and programming used by many universities and high schools around the world. Its conciseness, emphasis on vocabulary, and informal tone make it particularly appealing for readers with little or no experience. The book starts with the most basic programming concepts and gradually works its way to advanced object-oriented techniques.

In this fully updated and expanded edition, authors Allen Downey and Chris Mayfield introduce programming as a means for solving interesting problems.

Discover one concept at a time: tackle complex topics in a series of small steps with multiple examples Understand how to formulate problems, think creatively about solutions, and develop, test, and debug programs.

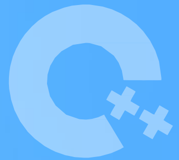
Learn about input and output, decisions and loops, classes and methods, strings and arrays, recursion and polymorphism Determine which program development methods work best for you and practice the important skill of debugging.

Free Copy available from:
https://greenteapress.com/wp/think-java-2e/

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

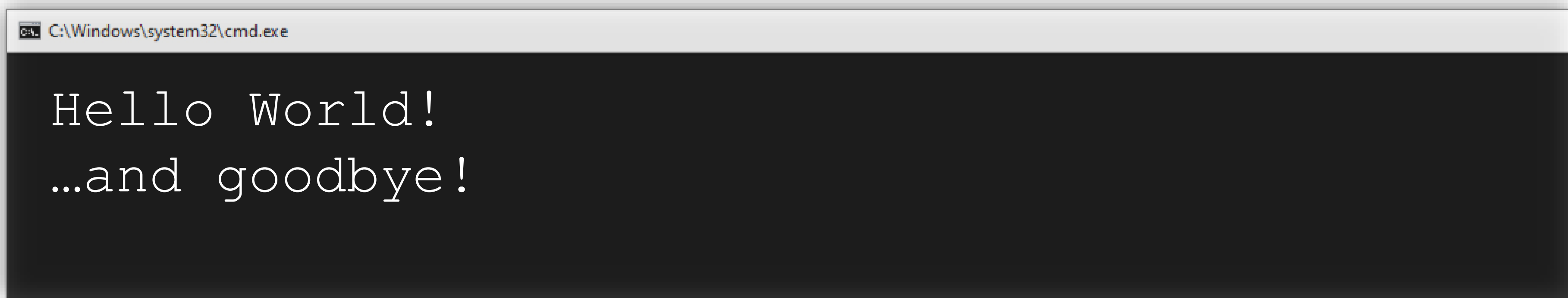
Ready to get started ?



Java Programs

```
1     public class Hello {
2           public static void main(String[] args) {
3                 System.out.println("Hello World!");
4                 System.out.println("..and goodbye!");
5           }
6     }
```

Compile
& Run



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Tools to get started

What tools do you need for creating Java programs?

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Tools to get started

- A Development Environment
- A Compiler
- Java Virtual Machine

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

- A Development Environment
- A Compiler
- Java Virtual Machine

A *development environment (or Integrated Development Environment - IDE)* helps you manage your code and provides convenient ways for you to write, compile, and run your code.

You don't actually *need* an integrated development environment:
All you need is software that lets you create simple text files, like notepad, or textpad.

- A Development Environment
- ➢ A Compiler
- Java Virtual Machine

A *compiler* takes the Java code (the simple text file) that you write and turns that code into a series of instructions called *bytecode*.

Humans can't readily compose or decipher bytecode instructions. But certain software that you run on your computer can interpret and carry out bytecode instructions.

- A Development Environment
- A Compiler
- ➢ Java Virtual Machine

You need a Java Virtual Machine (JVM).

A *Java Virtual Machine* is a piece of software. A Java Virtual Machine interprets and carries out bytecode instructions.

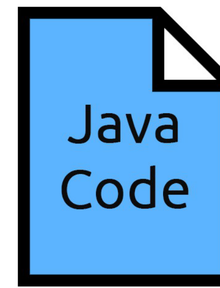
SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Development Kit (JDK) Editions

- Java Standard Edition (J2SE)
  - J2SE can be used to develop client-side standalone applications or applets.
- Java Enterprise Edition (J2EE)
  - J2EE can be used to develop server-side applications such as Java servlets, Java ServerPages, and Java ServerFaces.
- Java Micro Edition (J2ME).
  - J2ME can be used to develop applications for mobile devices such as cell phones.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

When you write a Java program, you write java code – a plain text file.



After writing the code, you run a program (that is, you apply a tool) to your source code. The program is a compiler.

The compiler translates your source code instructions into Java byte-code instructions.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

When you write a Java program, you write java code – a plain text file.

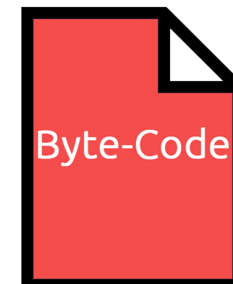
After writing the code, you run a program (that is, you apply a tool) to your source code. The program is a compiler.

The compiler translates your source code instructions into Java byte-code instructions.

When you write a Java program, you write java code – a plain text file.

After writing the code, you run a program (that is, you apply a tool) to your source code. The program is a compiler.

The compiler translates your source code instructions into Java byte-code instructions.



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Code

The code that a programmer creates is known as source code. The file that contains this must end in ".java", eg:

Hello.java

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Compiled Byte-Code

A class file is created as a result of successful compilation by the Java compiler from the ". java" file, eg:

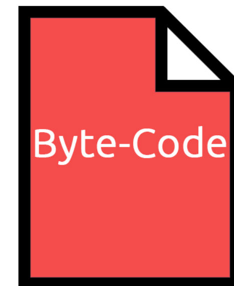
Hello.class

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Byte-Code

Byte-Code Portability!

With Java, you can take a bytecode file that you created with a Windows computer, copy the bytecode to, say, a Macintosh computer, and then run the byte-code with no trouble at all.

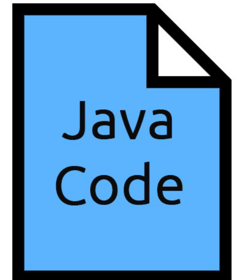


Hello.class

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A Simple Java Program

```
1       public class Hello {
2           public static void main(String[] args) {
3               System.out.println("Hello World!");
4           }
5       }
```

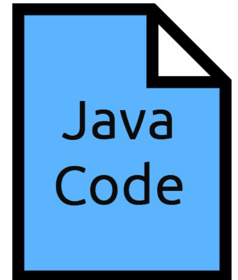


Hello.java

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A Simple Java Program

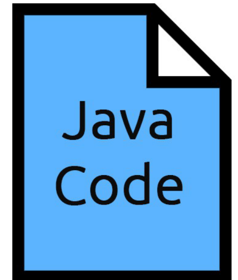
```
1    public class Hello {
2        public static void main(String[] args) {
3            System.out.println("Hello World!");
4        }
5    }
```



Hello.java

A Simple Java Program

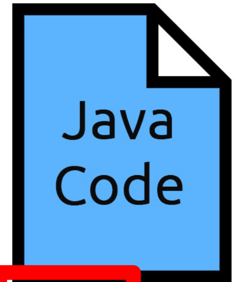
```
1       public class Hello {
2               public static void main(String[] args) {
3                       System.out.println("Hello World!");
4               }
5       }
```



Hello.java

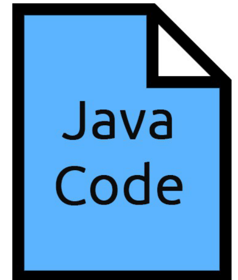
A Simple Java Program

```
1      public class Hello {
2          public static void main(String[] args) {
3              System.out.println("Hello World!");
4          }
5      }
```



A Simple Java Program

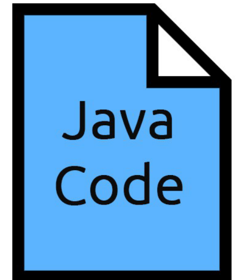
```
1     public class Hello {
2         public static void main(String[] args) {
3             System.out.println("Hello World!");
4         }
5     }
```



Hello.java

A Simple Java Program

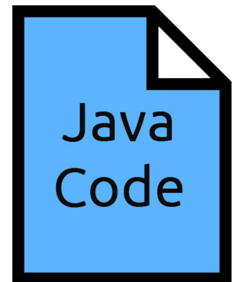
```
1    public class Hello {
2        public static void main(String[] args) {
3            System.out.println("Hello World!");
4        }
5    }
```



Hello.java

A Simple Java Program

```
1       public class Hello {
2               public static void main(String[] args) {
3                       System.out.println("Hello World!");
4               }
5       }
```

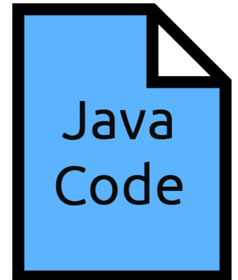


Hello.java

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A Simple Java Program

```
1     public class Hello {
2         public static void main(String[] args) {
3             System.out.println("Hello World!");
4         }
5     }
```

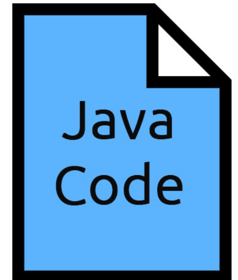


Hello.java

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A Simple Java Program

```
1       public class Hello {
2               public static void main(String[] args) {
3                       System.out.println("Hello World!");
4               }
5       }
```

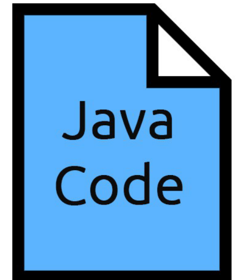


Hello.java

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A Simple Java Program

```
1    public class Hello {
2        public static void main(String[] args) {
3            System.out.println("Hello World!");
4        }
5    }
```

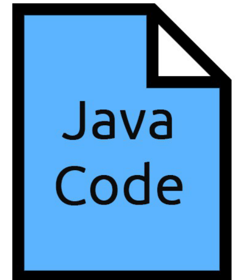


Hello.java

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

A Simple Java Program

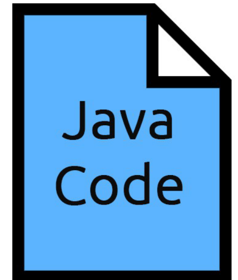
```
1     public class Hello {
2         public static void main(String[] args) {
3             System.out.println("Hello World!");
4         }
5     }
```



Hello.java

A Simple Java Program

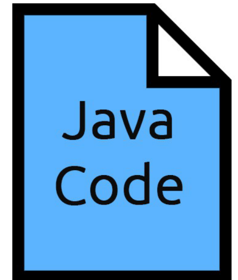
```
1   public class Hello {
2       public static void main(String[] args) {
3           System.out.println("Hello World!");
4       }
5   }
```



Hello.java

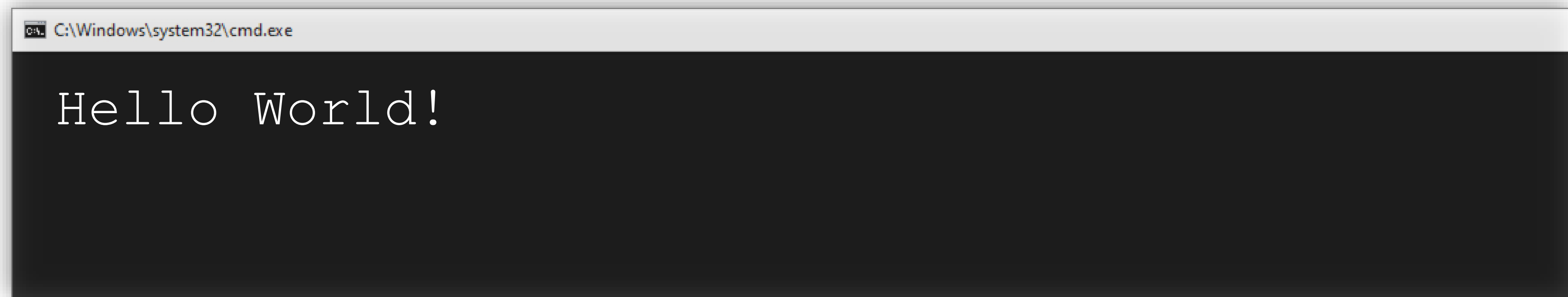
A Simple Java Program

```
1       public class Hello {
2           public static void main(String[] args) {
3               System.out.println("Hello World!");
4           }
5       }
```



Hello.java

Compile
& Run



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Programs – Shell Code

```
1       public class YourClassNameHere {
2           public static void main(String[] args)
3           {
4
5
8
9           }
10      }
11
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Fundamentals

Variables

A variable is a named storage location in the computer’s memory.

A literal is a value written into the code of a program.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Programs

```
1       public class JavaVariable
2       {
3               public static void main(String[] args)
4               {
5                       int myFirstNumber = 5;
8
9                       System.out.println(myFirstNumber);
10              }
11      }
```

Compile & Run

```
C:\Windows\system32\cmd.exe
5
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java
Fundamentals

Data Types

Data Types are the types of data that can be stored in a variable.

Java Primitive Data Types (Complete)

Data Type	*Size*	*Range*
byte	1 byte	Integers (-128 to +127)
short	2 bytes	Integers (-32,768 to +32,767)
int	4 bytes	Integers (-2,147,483,648 to +2,147,483,647)
long	8 bytes	Integers (-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)
float	4 bytes	Floating-point numbers (±3.4E-38 to ±3.4E38, 7 digits of accuracy)
double	8 bytes	Floating-point numbers (±1.7E-308 to ±1.7E308, 15 digits of accuracy)
char	2 bytes	Stores a single character/letter or ASCII values
boolean	1 bit	Stores either true or false

Primitive Data Types

There are many different types of data. Variables are classified according to their data type. The data type determines the kind of data that may be stored in them.

The data type also determines the amount of memory the variable uses, and the way the variable formats and stores data.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Primitive Data Types - Declaration

```
byte apples;
short minutes;
int temp;
long days;

apples = 20;
minutes = 200;
temp = -3;
days = 182500L;
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Integer Data Types

When you declare a <u>long</u> data type, the value must have a suffix of the letter L.
Example: 120L would be treated as a *long*.

```
long myCounter;

myCounter = 120;
```

Java will assume an ***int*** type

```
long myCounter;

myCounter = 120L;
```

Forced to be of type ***long***

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Primitive Data Types - Declaration

```
double myDouble;
float myFloat;
```

```
myDouble = 123456.789456;
myFloat = 200.123f;
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Floating-Point Data Types

The floating-point data types include *float* and *double*.

When you write a floating-point literal in your program code, Java assumes it to be of the *double* data type. You can force a floating-point literal to be treated as a *float* by suffixing it with the letter F. Example: 14.0F would be treated as a *float*.

```
float pay;
pay = 1800.99;
```

This statement will give an error message (1800.99 seen as a **double**)

```
float pay;
pay = 1800.99F;
```

Forced to be of type **float**

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Primitive Data Types - Declaration

```
char myChar;
boolean myBool;


myChar = 'a';
myBool = true;
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

String Data Type - Declaration

```
String myText;
```

```
myText = "Hello World!";
```

Java
Fundamentals

Strings

A string is a sequence of characters and is a data type used to represent text rather than numbers. It is comprised of a set of characters that can also contain spaces and numbers.

Strings in Java - Declaration

```
String myText;
myText = "Hello World!";
```

9 different data types:

4 x integers (byte, short, int and long)

2 x floating point (float and double)

char

String

boolean

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java
Fundamentals

Operators

Operators are used to perform operations on variables and values.

Arithmetic Operators

Java offers a multitude of operators for manipulating data.

Most of its operators can be divided into the following four groups:

arithmetic, bitwise, relational, and logical.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Arithmetic Operators

Addition (+)

The addition operator returns the sum of its two operands.

```
answer = 2 + 8;         // assigns 10 to variable answer

pay = salary + bonus;   // assigns the value of salary + bonus to variable pay

number = number + 3;    // assigns number + 3 to variable number
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Arithmetic Operators

Subtraction (-)

The subtraction operator returns the value of its right operand subtracted from its left operand.

```
answer = 5 - 4;        // assigns 1 to variable answer

pay = salary - tax;    // assigns the value of salary - tax to variable pay

number = number - 1;   // assigns number - 1 to variable number
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Arithmetic Operators
Multiplication (*)

The multiplication operator returns the product of its two operands.

```
answer = 5 * 4;         // assigns 20 to variable answer

pay = hours * rate      // assigns the value of (hours * rate) to variable pay

students  = students * 2;   // assigns (students * 2) to variable students
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Arithmetic Operators

Division (/)

The division operator returns the quotient of its left operand divided by its right operand.

```
answer = 20 / 4;              // assigns 5 to variable answer

average = marks / number;     // assigns the value of (marks / number) to
                              //average variable

half  = number / 2;           // assigns (number / 2) to variable half
```

Integer division

When both operands of a division statement are integers, the statement will result in ***integer division***. This means that the result of the division will be an integer as well. If there is a remainder, it will be discarded.

```
double parts;

parts = 22 / 4;   // will assign the value of 5.0
```

In order for a division operation to return a floating-point value, one of the operands must be of a floating-point data type.

```
double parts;

parts = 22.0 / 4;   // will assign the value of 5.5
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Arithmetic Operators
Modulus (%)

The modulus operator returns the remainder of a division operation involving two integers.

```
leftOver = 22 / 4;        // assigns 2 to variable leftOver
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Operator Precedence

Operator precedence refers to the rules for the order in which parts of a mathematical expression are evaluated.

The multiplication, division, and remainder operators have the same precedence, and it is higher than the precedence of the addition and subtraction operators.

Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators.

Precedence rules can be overridden by explicit parentheses.

https://introcs.cs.princeton.edu/java/11precedence/

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Operator Precedence

$3 + 6 / 2 = ?$

$3 + 6 / 2 = 4.5$

or

$3 + 6 / 2 = 6$

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Explicit Parenthesis example

(3 + 6) / 2

10 / (5+6)

15 * ((6+5)-7)

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Fundamentals

Concatenation

Concatenation is the operation of *joining* character strings end-to-end. For example, the concatenation of "snow" and "ball" is "snowball".

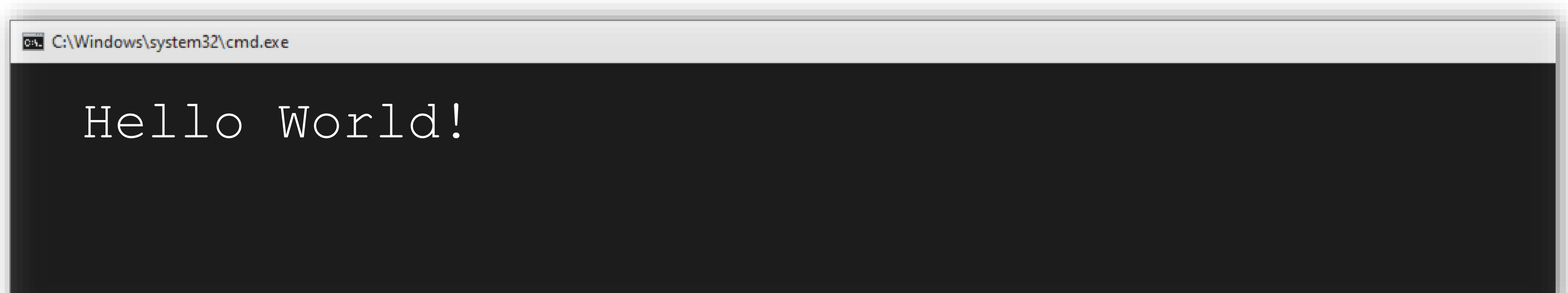
Displaying Multiple Items with the + Operator

When the + operator is used with strings ,we call it a *string concatenation operator*. To concatenate means to append. The string concatenation operator appends one string to another.

```
System.out.println("Hello " + "World!");
```

The + operator produces a string that is a combination of the 2 strings both sides of the operator.

Compile & Run



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

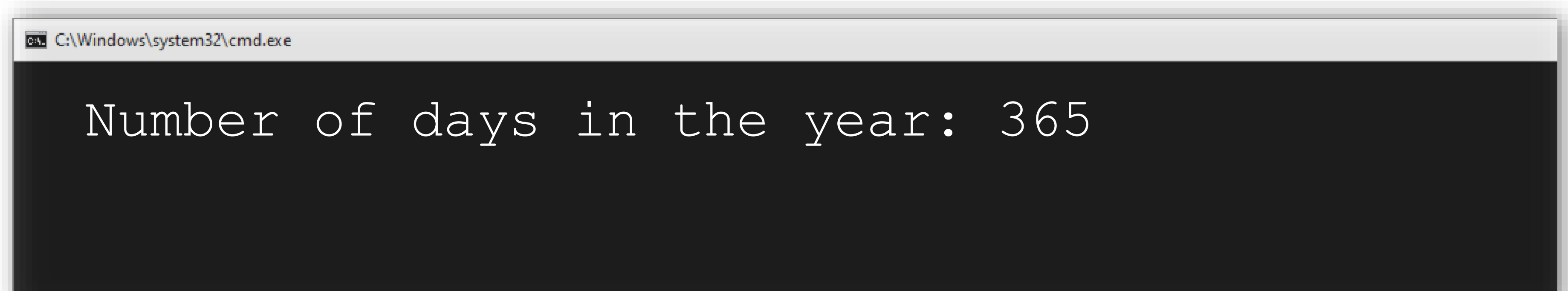
Displaying Multiple Items with the + Operator

We can also use the + operator to concatenate the contents of a variable to a string.

```
myVar = 365;
System.out.println("Number of days in the year:  " + myVar);
```

The + operator is used to concatenate the contents of the ***number*** variable with the string "***Number of days in the year:*** ". The **+** operator converts the ***myVar*** variable's value from an integer to a string and then appends the new value.

Compile & Run



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Fundamentals

Keeping your code tidy
Use comments and indentation to improve readability

Use descriptive names for Classes and Variables

Comments

Comments are ***notes of explanation*** that document lines or sections of a program. Comments are part of the program, but the ***compiler ignores them***. They are intended for people who may be reading the source code.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Comments

Three ways to comment in Java

Single-Line comments (//)

Multi-line comments (/*….. */)

Documentation comments (/**….. */)

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Comments

Single-Line comment

You simply place two forward slashes (//) where you want the comment to begin. The compiler ignores everything from that point to the end of the line.

```
1   // This is a single-line comment
2
3   public class …
4   {
5         ….
6   }
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Comments

Multi-Line comment

Multi-Line comments start with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/). Everything between these markers is ignored.

```
1   /*
2         This is a
3         Multi-Line comment
4   */
5   public class …
6   {
7         …
8   }
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Comments

Documentation Comments

Documentation comments starts with /** and ends with */. Normally you write a documentation comment just before ***class*** and ***method headers***, giving a brief description of the ***class*** or ***method***.

These comments can be read and processed by a program named ***javadoc***, which comes with the Sun JDK. The purpose of the ***javadoc*** program is to read Java source code files and generate attractively formatted HTML files that document the source code.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Comments

Documentation Comments

```
1   /**
2         This class creates a program that calculates
3         company payroll
4   */
5   public class Comment
6   {
7         /**
8                     The main method is the program's starting point
9         */
10        public static void main(String[] args)
11        {
12                    ...
13        }
14  }
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Indentation

Indentation is a fundamental aspect of code styling and plays a large role in influencing readability. Indented code is easier to read through than un-indented code.

Proper code indentation will ensure your code is:

Easier to read
Easier to understand
Easier to modify
Easier to maintain
Easier to enhance

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Identifiers

Variable names and class names are examples of identifiers (represents some element of a program)

You should always choose names for your variables that give an indication of what they are used for:

```
int y;
```

This gives us no clue as to what the purpose of the variable is.

```
int numberOfBikes;
```

```
int numberofbikes;
```

numberOfBikes gives anyone reading the program an idea of what the variable is used for.

Identifiers

The following rules must be followed with all identifiers:

- The first character must be one of the letters **a-z**, **A-Z**, underscore ”_”, or the dollar sign “**$**”
- After the first character, you may use the letters **a-z**, **A-Z**, underscore ”_”, the dollar sign “**$**”, and the digits 0-9
- No spaces
- Uppercase and lowercase characters are distinct. This means that numberOfBikes is not the same as numberofbikes.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Variable and Class names

Variable

Start variable names with a lowercase letter

Each subsequent word's first letter must be capitalised

Example:

numberOfBikes

Class

Start class names with an uppercase letter

Each subsequent word's first letter must be capitalised

Example:

CityCars

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Fundamentals

Reading Keyboard Input using Scanner

Reading Keyboard Input

The Scanner class

Objects of the ***Scanner*** class can be used to read input from the keyboard.

The Java API has an object ***System.in*** which refers to the standard input device (normally the keyboard). The ***System.in*** object reads input only as byte values which isn't very useful. To work around this, we use the ***System.in*** object in conjunction with an object of the ***Scanner*** class.

The ***Scanner*** class is designed to read input from a source (***System.in***) and provides methods that you can use to retrieve the input formatted as ***primitive values*** or ***strings***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Scanner class : *import* statement

The ***Scanner*** class is not automatically available to your Java programs. Any program that uses the ***Scanner*** class should have the following statement near the beginning of the file, before any class definition:

```
import java.util.Scanner;
```

This statement tells the Java compiler where in the Java library to find the ***Scanner*** class, and makes it available to your program.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

The Scanner class

First, you create a ***Scanner*** object and connect it to the ***System.in*** object:

```
Scanner keyboard
```

Reading Keyboard Input

The Scanner class

First, you create a ***Scanner*** object and connect it to the ***System.in*** object:

```
Scanner keyboard = new Scanner(System.in);
```

Declares a variable named ***keyboard.***
The data type of the variable is ***Scanner***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Scanner class methods

The Scanner class has methods for reading ***strings***, ***bytes***, ***int***egers, ***long*** integers, ***short*** integers, ***floats*** and ***double***s.

Reading Keyboard Input

Scanner class methods : nextInt

Returns input as an ***int***

```
1   int number;
2   Scanner keyboard = new Scanner(System.in);
3   System.out.println("Enter a integer value: ");
4   number = keyboard.nextInt();
```

The ***nextInt*** method formats the input that was entered at the keyboard as an ***int***, and then returns it.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Scanner class methods : nextDouble

Returns input as a ***double***

```
1   double number;
2   Scanner keyboard = new Scanner(System.in);
3   System.out.println("Enter a double value: ");
4   number = keyboard.nextDouble();
```

The ***nextDouble*** method formats the input that was entered at the keyboard as a ***double***, and then returns it.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Scanner class methods : nextByte

Returns input as a ***byte***

```
1   byte x;
2   Scanner keyboard = new Scanner(System.in);
3   System.out.println(“Enter a byte value: “);
4   x = keyboard.nextByte();
```

The ***nextByte*** method formats the input that was entered at the keyboard as a ***byte***, and then returns it.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Scanner class methods : nextFloat

Returns input as a ***float***

```
1    float number;
2    Scanner keyboard = new Scanner(System.in);
3    System.out.println("Enter a float value: ");
4    number = keyboard.nextFloat();
```

The ***nextFloat*** method formats the input that was entered at the keyboard as a ***float***, and then returns it.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Scanner class methods : nextLong

Returns input as a ***long***

```
1   long number;
2   Scanner keyboard = new Scanner(System.in);
3   System.out.println("Enter a long value: ");
4   number = keyboard.nextLong();
```

The ***nextLong*** method formats the input that was entered at the keyboard as a ***long***, and then returns it.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

```
1    import java.util.Scanner;  // Needed for the Scanner class
2
3    public class InputProblem
4    {
5        public static void main(String[] args)
6        {
7
8                int age;
9
10
11               Scanner keyboard = new Scanner(System.in);
12
13               System.out.print("What is your age?");
14               age = keyboard.nextInt();
15
16               System.out.println("Your age is " + age);
18       }
19   }
```

Variable declarations

Create Scanner object to read input

Get user's age

C:\Windows\system32\cmd.exe

```
What is your age? 25
Your age is 25
Press any key to continue . . .
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

```
1    import java.util.Scanner;  // Needed for the Scanner class
2
3    public class InputProblem
4    {
5        public static void main(String[] args)
6        {
7
8            int age;
9            int yearOfBirth;
10
11           Scanner keyboard = new Scanner(System.in);
12
13           System.out.print("What is your age?");
14           age = keyboard.nextInt();
15           System.out.print("What is your year of birth?");
16           yearOfBirth = keyboard.nextInt();
17
18           System.out.println("You're " + age + ", you were born in " + yearOfBirth)
19       }
20   }
```

Variable declarations

Create Scanner object to read input

Get user's age

Get user's year of birth

C:\Windows\system32\cmd.exe

```
What is your age? 25
What is your year of birth?  1995
You're 25, you were born in 1995
Press any key to continue . . .
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

Printed on the computer screen when application runs:

What is your age? ***25 [enter]***

What is your annual income? ***80000 [enter]***

What is your name? Hello .Your age is 25 and your income is R80000.00

The program does not give the user time to enter his/her name

Problem!!

```
11                Scanner keyboard = new Scanner(System.in);

                  ...t.print("What is your age?   ");
                  ...oard.nextInt();

                  System.out.print("What is your annual income?  ");
                  income = keyboard.nextDouble();
18
19                System.out.print("What is your name?  ");
20                name = keyboard.nextLine();
21
22                System.out.println("Hello " + name + ". Your age is " +
23                        age + " and your income is R" + income);
24        }
25    }
```

Reading Keyboard Input

When the user types keystrokes at the keyboard, those keystrokes are stored in an area of memory called the ***keyboard buffer***.

Pressing the "***Enter***" key causes a ***new-line*** character to be stored in the keyboard buffer.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

keyboard buffer

25
/n

nextInt() method

age =

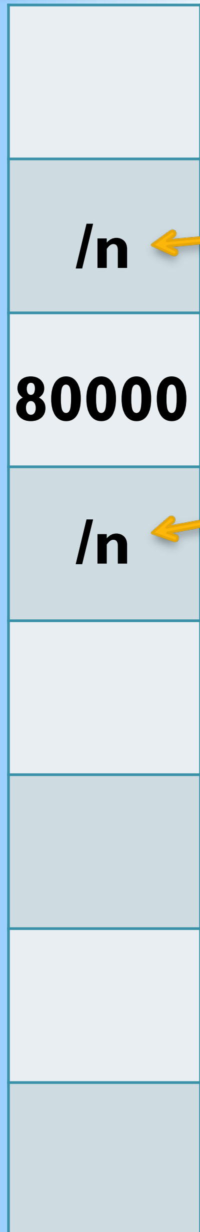
Stops when it sees the new-line character

keyboard buffer, and then stopped when it encountered the **newline** character. The newline character was not read and remained in the keyboard buffer.

```
13
14        age = keyboard.nextInt();
15
16        System.out.print("What is your annual income?  ");
17        income = keyboard.nextDouble();
18
19        System.out.print("What is your name?  ");
20        name = keyboard.nextLine();
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input



nextDouble() method

Skips newline character

income =

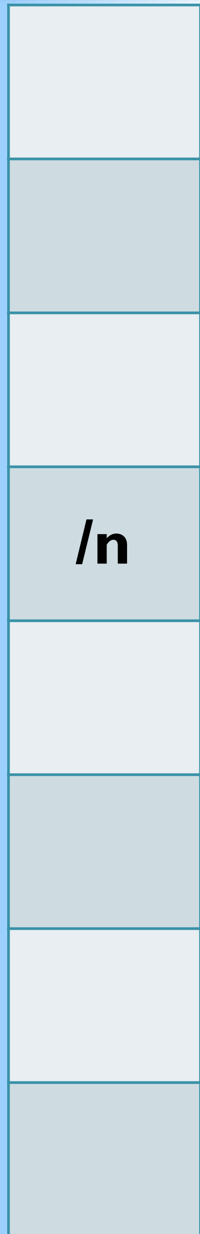
Stops when it sees the newline character

newline characters it encounters. It skips the newline character, reads the value 80000.00 and stops reading when it encounters the newline character which is then left in the keyboard buffer.

```
16        System.out.print("What is your annual income?  ");
17        income = keyboard.nextDouble();
18
19        System.out.print("What is your name?  ");
20        name = keyboard.nextLine();
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input



Next the user was asked to enter his/her ***name***

In line 20 the **nextLine()** method is called.

The ***nextLine()*** method, however, is not designed to skip over an initial newline character. If a newline character is the first character that ***nextLine()*** method encounters, then nothing will be read. It will immediately terminate and the user will not be given a chance to enter his or her name.

```
19        System.out.print("What is your name?  ");
20        name = keyboard.nextLine();
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Reading Keyboard Input

```
1     import java.util.Scanner;  // Needed for the Scanner class
2
3     public class InputProblem
4     {
5           public static void main(String[] args)
6
7
8
9
10
11
12
13                      System.out.print("What is your age?   ");
14                      age = keyboard.nextInt();
15
16                      System.out.print("What is your annual income?  ");
17                      income = keyboard.nextDouble();
18                      keyboard.nextLine();
19                      System.out.print("What is your name?  ");
20                      name = keyboard.nextLine();
21
22                      System.out.println("Hello " + name + ". Your age is " +
23                                   age + " and your income is R" + income);
24          }
25    }
```

The purpose of this call is to consume, or remove, the newline character that remains in the keyboard buffer. We do not need to keep the method's return value so we do not assign the method's return value to any variable

Java Fundamentals

Decision Structures

The `if` Statement

- The `if` statement decides whether a section of code executes or not.
- The `if` statement uses a `boolean` value to decide whether the next statement or block of statements executes.

if (boolean expression is true)

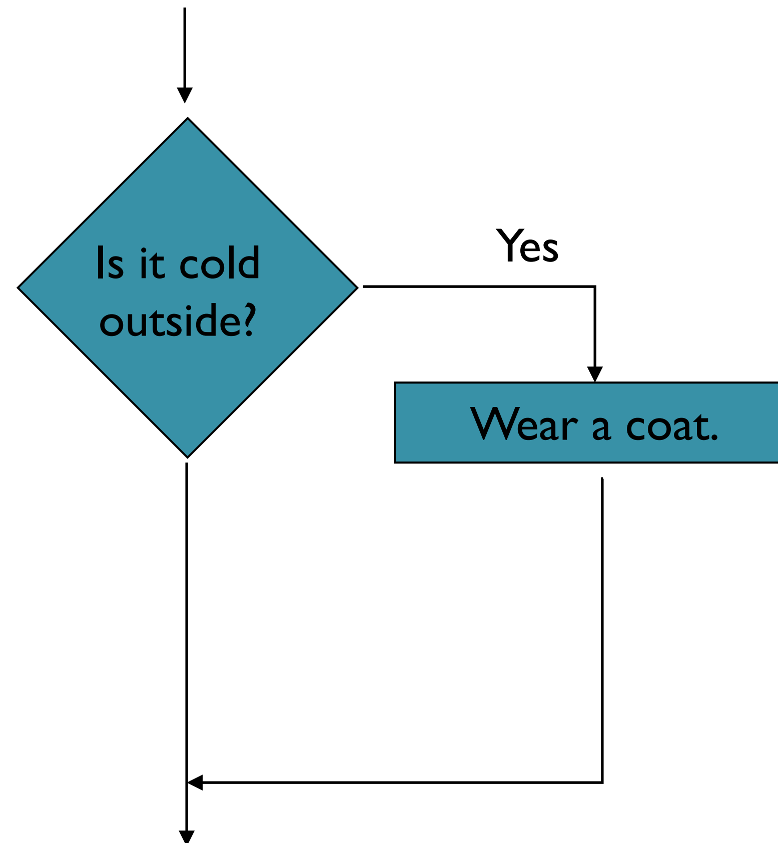
execute next statement.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Flowcharts

- ***If statements*** can be modeled as a flow chart.

```
if (coldOutside)
   wearCoat();
```

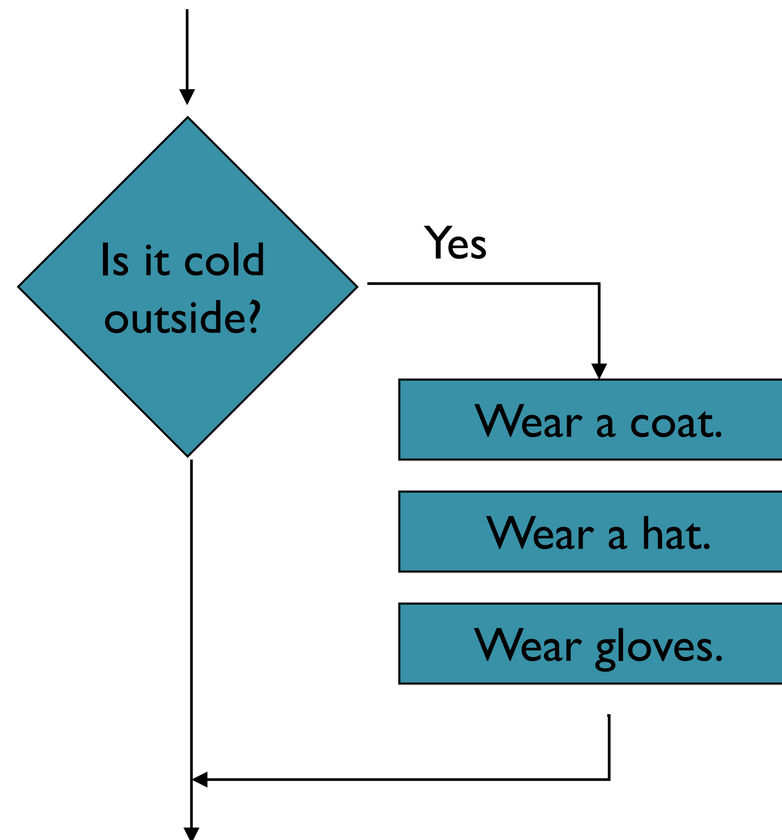


SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Flowcharts

- A block ***`if` statement*** may be modeled as:

```
if (coldOutside)
{
   wearCoat();
   wearHat();
   wearGloves();
}
```



Note the use of curly braces to block several statements together.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Relational Operators

- In most cases, the `boolean` expression, used by the `if` statement, uses *relational operators*.

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

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Boolean Expressions

- A *boolean expression* is any variable or calculation that results in a *true* or *false* condition.

Expression	Meaning
x > y	Is x greater than y?
x < y	Is x less than y?
x >= y	Is x greater than or equal to y?
x <= y	Is x less than or equal to y.
x == y	Is x equal to y?
x != y	Is x not equal to y?

Programming Style and `if` Statements

- An `if` statement can span more than one line; however, it is still one statement.

```
if (average > 95)
   grade = 'A';
```

is functionally equivalent to

```
if(average > 95) grade = 'A';
```

Programming Style and `if` Statements

- Rules of thumb:
  - The conditionally executed statement should be on the line *after* the `if` condition.
  - The conditionally executed statement should be indented one level from the `if` condition.
  - If an `if` statement does not have the block curly braces, it is ended by the first semicolon encountered after the `if` condition.

```
if (expression)      ← No semicolon here.
  statement;         ← Semicolon ends statement here.
```

Block `if` Statements

- Conditionally executed statements can be grouped into a block by using curly braces { } to enclose them.
- If curly braces are used to group conditionally executed statements, the `if` statement is ended by the closing curly brace.

```
if (expression)
{
  statement1;
  statement2;
}
```

Curly brace ends the statement.

Comparing Characters

- Characters can be tested with relational operators.
- Characters are stored in memory using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are *ordinal*, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

```
char c = 'A';
if(c < 'Z')
  System.out.println("A is less than Z");
```

Flags

- A flag is a `boolean` variable that monitors some condition in a program.
- When a condition is true, the flag is set to `true`.
- The flag can be tested to see if the condition has changed.

```
if (average > 95)
  highScore = true;
```

- Later, this condition can be tested:

```
if (highScore)
  System.out.println("That's a high score!");
```

Comparing Characters

- Characters can be tested with relational operators.
- Characters are stored in memory using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are *ordinal*, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

```
char c = 'A';
if(c < 'Z')
  System.out.println("A is less than Z");
```

`if` Statements

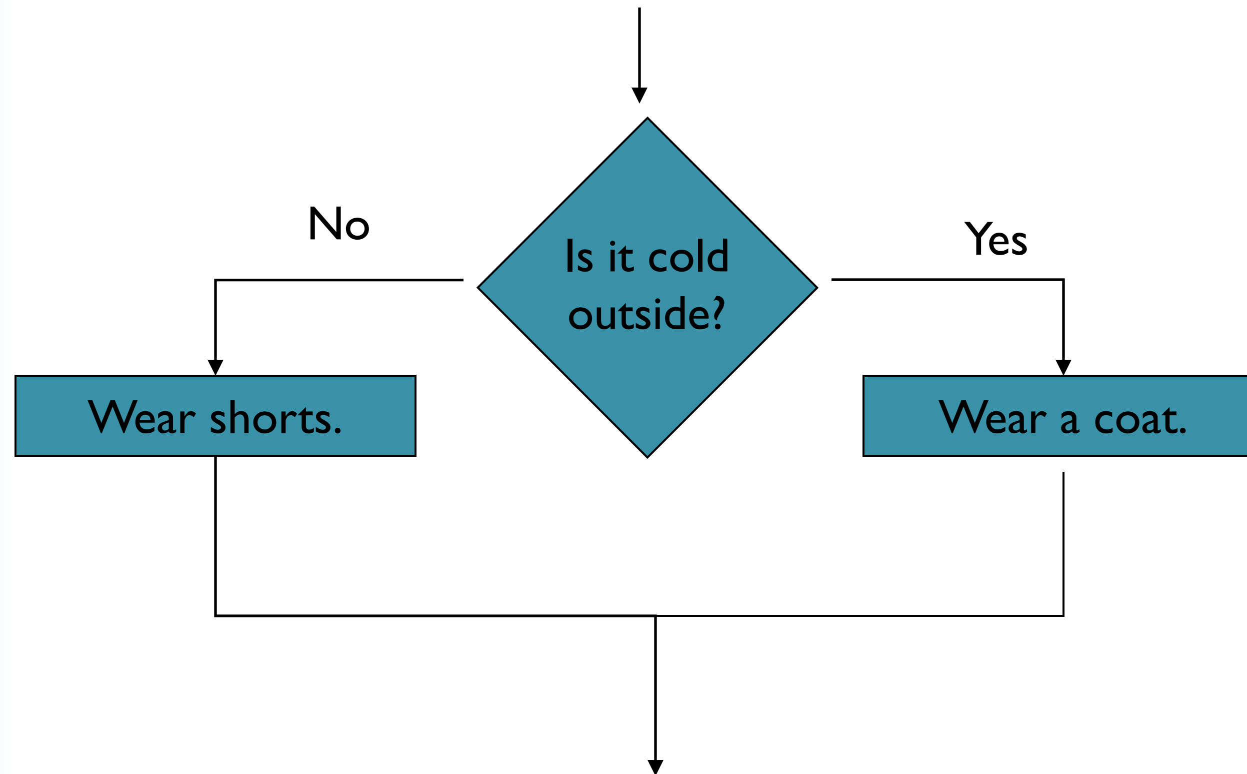
```
if (coldOutside == true){
        System.out.println(“Wear Jacket”);
}
```

`if-else` Statements

- The `if-else` statement adds the ability to conditionally execute code when the `if` condition is **false**.

```
if (expression)
  statementOrBlockIfTrue;
else
  statementOrBlockIfFalse;
```

`if-else` Statement Flowcharts



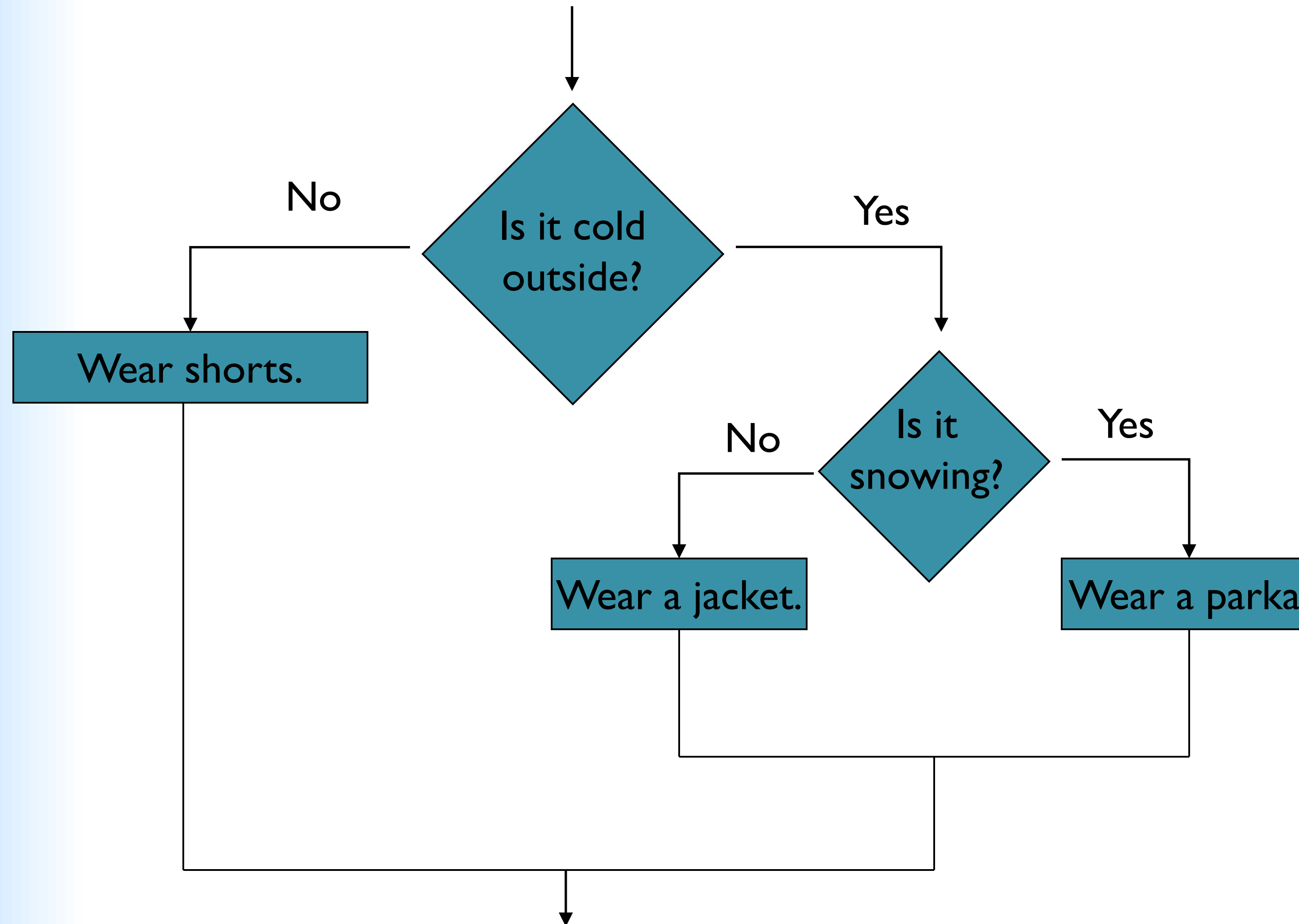
if else Statements

```
if (coldOutside == true){
    System.out.println(“Wear Jacket”);
}else{
    System.out.println(“Wear shorts”);
}
```

Nested `if` Statements

- When an `if` statement appears inside another `if` statement (single or block) it is called a *nested* `if` statement.
- The nested `if` is executed only if the outer `if` statement results in a true condition.

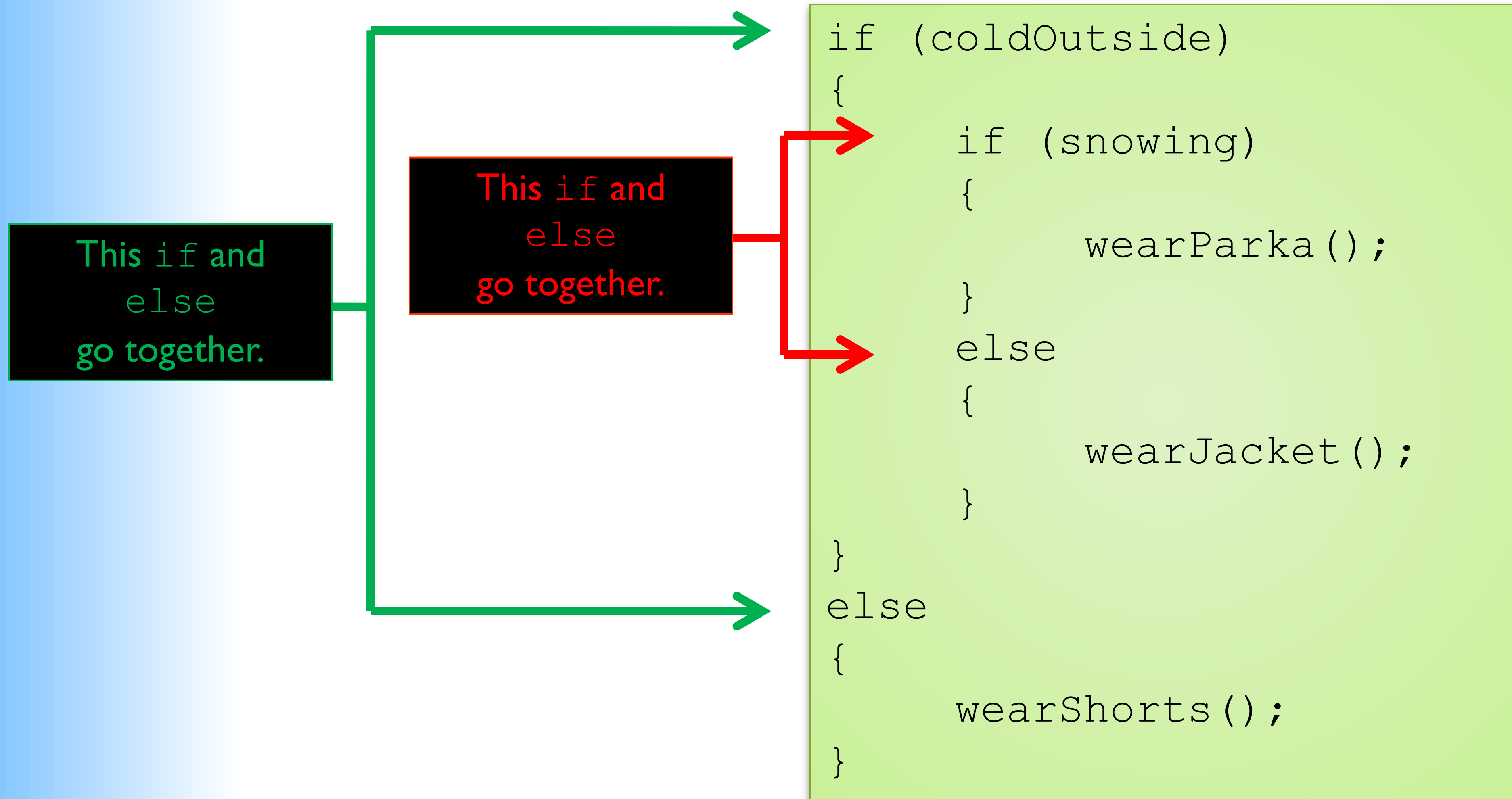
Nested `if` Statement Flowcharts



Nested `if` Statements

```
if (coldOutside == true){
      if (snowing == true){
            System.out.println("Wear Parka");
         }else {
          System.out.println("Wear Jacket");
         }
}else{
     System.out.println("Wear shorts");
}
```

Alignment and Nested `if` Statements



`if-else-if` Statements

- Nested `if` statements can become very complex.
- The `if-else-if` statement makes certain types of nested decision logic simpler to write.

`if-else` Matching

- Curly brace use is not required if there is only one statement to be conditionally executed.
- **However, curly braces make the program more readable.**
- Additionally, proper indentation makes it much easier to match up else statements with their corresponding `if` statement.

if-else-if Statements

```
if (expression_1)
{
   statement;
   statement;
   etc.
}
else if (expression_2)
{
   statement;
   statement;
   etc.
}
```

If `expression_1` *is true these statements are executed, and the rest of the structure is ignored.*

Otherwise, if `expression_2` *is true these statements are executed, and the rest of the structure is ignored.*

Insert as many else if clauses as necessary

```
else
{
   statement;
   statement;
   etc.
}
```

These statements are executed if none of the expressions above are true.

Logical Operators

- Java provides two binary *logical operators* (`&&` and `||`) that are used to combine `boolean` expressions.
- Java also provides one *unary* (`!`) logical operator to reverse the truth of a `boolean` expression.

Logical Operators

<table>
<tr><th>Operator</th><th>Meaning</th><th>Effect</th></tr>
<tr><td>&&</td><td>AND</td><td>Connects two boolean expressions into one. Both expressions must be true for the overall expression to be true.</td></tr>
<tr><td>||</td><td>OR</td><td>Connects two boolean expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which one.</td></tr>
<tr><td>!</td><td>NOT</td><td>The ! operator reverses the truth of a boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.</td></tr>
</table>

The `&&` Operator

- The logical AND operator (`&&`) takes two operands that must both be `boolean` expressions.
- The resulting combined expression is true if (and *only* if) both operands are true.

Expression 1	Expression 2	Expression1 && Expression2
true	false	false
false	true	false
false	false	false
true	true	true

The `||` Operator

- The logical OR operator (`||`) takes two operands that must both be `boolean` expressions.
- The resulting combined expression is false if (and *only* if) both operands are false.

<table>
<tr><th>Expression 1</th><th>Expression 2</th><th>Expression1 || Expression2</th></tr>
<tr><td>true</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>true</td><td>true</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
</table>

The | | Operator

```
if( salary >=30000 ||  yearsOnJob >= 2)
{
        System.out.println(“You qualify for the loan!”);
}

else
{
        System.out.println(“You do not qualify for the loan”);
}
```

The ! Operator

- The ! operator performs a logical NOT operation.
- If an *expression* is true, !*expression* will be false.

```
if (!(temperature > 100))
   System.out.println("Below the maximum temperature.");
```

- If `temperature > 100` evaluates to false, then the output statement will be run.

Expression 1	!Expression1
true	false
false	true

Order of Precedence

- The `!` operator has a higher order of precedence than the `&&` and `||` operators.
- The `&&` and `||` operators have a lower precedence than relational operators like < and >.
- Parenthesis can be used to force the precedence to be changed.

Checking numeric ranges

- When determining whether a number is inside a range, it's best to use the && operator.

Range 20 to 40

```
if ( x >= 20 && x <=40 )
      System.out.println(x + " is in the range");
```

Checking numeric ranges

- When determining whether a number is outside a range, it's best to use the || operator.

Outside range 20 to 40

```
if ( x < 20 || x > 40 )
      System.out.println(x + " is outside the range");
```

Comparing `String` Objects

- In most cases, you cannot use the relational operators to compare two `String` objects.
- Reference variables contain the address of the object they represent.
- Unless the references point to the same object, the relational operators will not return true.

String name1 = “Thandi”;

String name2 = “Joseph”;

if (name1 == name2)

This statement will be false

Comparing `String` Objects

- In most cases, you cannot use the relational operators to compare two `String` objects.
- Reference variables contain the address of the object they represent.
- Unless the references point to the same object, the relational operators will not return true.

String name1 = "Thandi"; String name2 = "Thandi";

if (name1 == name2) This statement will be false as well

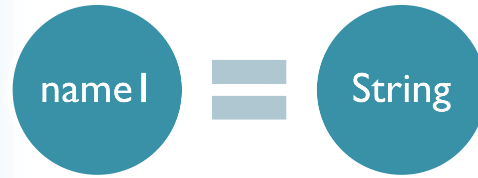
Comparing `String` Objects

String name1 = “Thandi”;

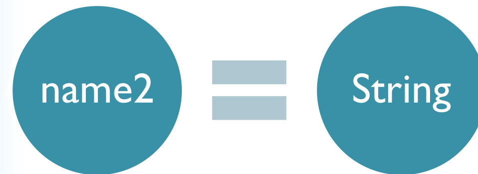
String name2 = “Thandi”;

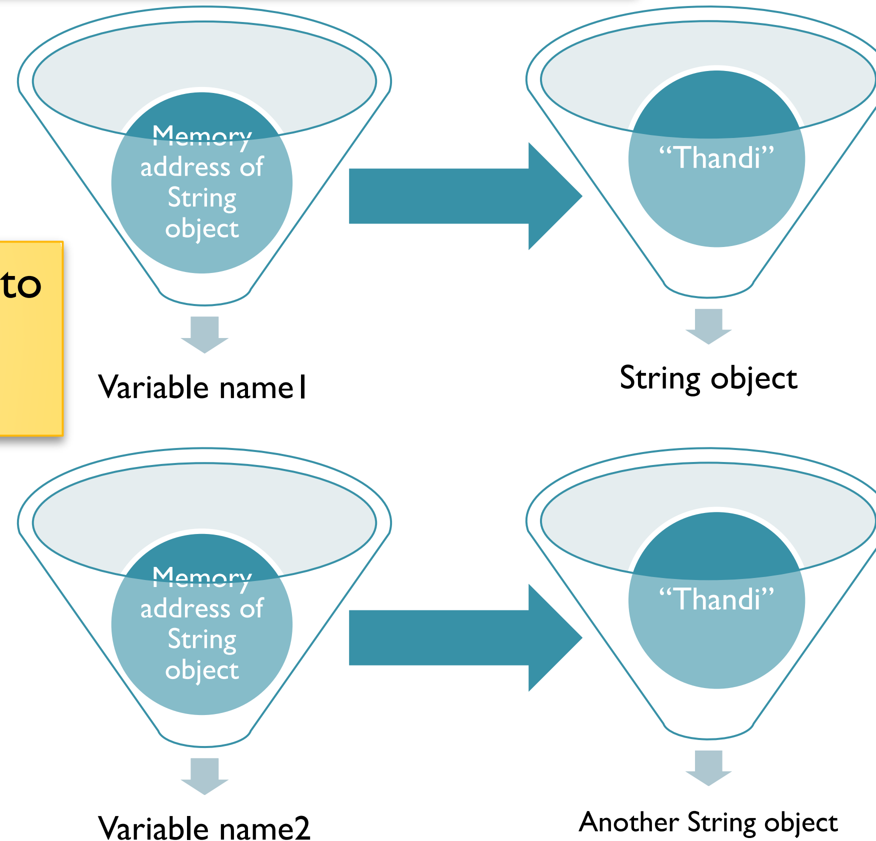
if (name1 == name2)

This statement will be false as well



Because ***name1*** and ***name2*** does not point to the same object, they are not the same according to the relational operators





Comparing `String` Objects

The `equals` method

String name1 = "Thandi";

String name2 = "Thandi";

if (name1.equals(name2))

This statement will be true

if (name1.equals("Thandi"))

This statement will be true

if (!name1.equals("Thandi"))

This statement will be false

Ignoring Case in String Comparisons

- In the `String` class the `equals` methods is case sensitive.
- In order to compare two `String` objects that might have different case, use:
  - `equalsIgnoreCase`

```
if (name1.equalsIgnoreCase(name2)){
        System.out.print("The names are the same");
}
```

The *String* Class

String Methods

Because the ***String*** type is a ***class*** instead of a ***primitive data type***, it provides numerous ***methods*** for working with strings.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The *String* Class

charAt() Method

This method returns the ***character*** at the specified ***position***.

```
char letter;
String name = "Arnold";
letter = name.charAt(2);
```

0	1	2	3	4	5
A	***r***	***n***	***o***	***l***	***d***

After this code executes, the variable ***letter*** will hold the character ***'n'***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The ***String*** Class

length() Method

This method returns the ***number of characters*** in a string.

```
int stringSize;
String name = “Arnold”;
stringSize = name.length();
```

0	1	2	3	4	5
A	***r***	***n***	***o***	***l***	***d***

After this code executes, the variable ***stringSize*** will hold the value ***6***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The *String* Class

toLowerCase() Method

This method returns a new string that is the ***lowercase*** equivalent of the string contained in the calling object.

```
String bigName = "ARNOLD";
String littleName = bigName.toLowerCase();
```

After this code executes, the variable ***littleName*** will hold the string ***"arnold".***

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The ***String*** Class

toUpperCase() Method

This method returns a new string that is the ***uppercase*** equivalent of the string contained in the calling object.

```
String littleName = “arnold”;
String bigName = littleName.toUpperCase();
```

After this code executes, the variable ***bigName*** will hold the string ***“ARNOLD”.***

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Pseudocode

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Pseudocode

Pseudocode is a detailed description of what a computer program must do, expressed in an English like language rather than in a programming language.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Pseudocode Convention

- Statement are written in simple English
- Each instruction is written on a separate line
- Keywords and indentation are used to signify particular control structures.
- Each set of instructions is written from top to bottom, with only one entry and one exit.
- Groups of statements may be formed into modules, and that group given a name.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Levels of Program Development

1. Define the problem. →Human thought
2. Plan the problem solution. → writing the algorithm [pseudo-natural language (English)
3. Code the program. → High Level Programming Language (Java)
4. Compile the program. → Machine Code
5. Run the program.
6. Test and debug the program.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Pseudocode Example

Write a Program to Print the Sum of two integer Numbers

- Start the program
- Read the first number and save in the variable (N1)
- Read the second number and save in the variable (N2)
- Save the sum of both numbers in the variable Sum
  Sum = N1 + N2
- Print the variable (Sum)
- End the program

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Pseudocode Example

Pseudocode:

- Start the program
- Create a variable to hold a counter from 2 to 30.
- Initialize the counter to 2.
- Create a variable to hold the sum.
- Initialize the sum to zero.
- Loop While the counter is less-than-or-equal to 30
  - add the counter to the sum
  - add two to the counter.
- repeat until the counter reach 30
- Print the sum.
- End of program

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

When planning for a problem solution, algorithms are used to outline the solution steps using**English like statements**, called ***pseudocode***.

or

A ***flowchart*** , which is a graphical representation of an algorithm.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Fundamentals

The Switch Statement

The `switch` Statement

- The `if-else` statement allows you to make true / false branches.
- The `switch` statement allows you to use a value to determine how a program will branch.
- The `switch` statement can evaluate a variable and make decisions based on the value.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

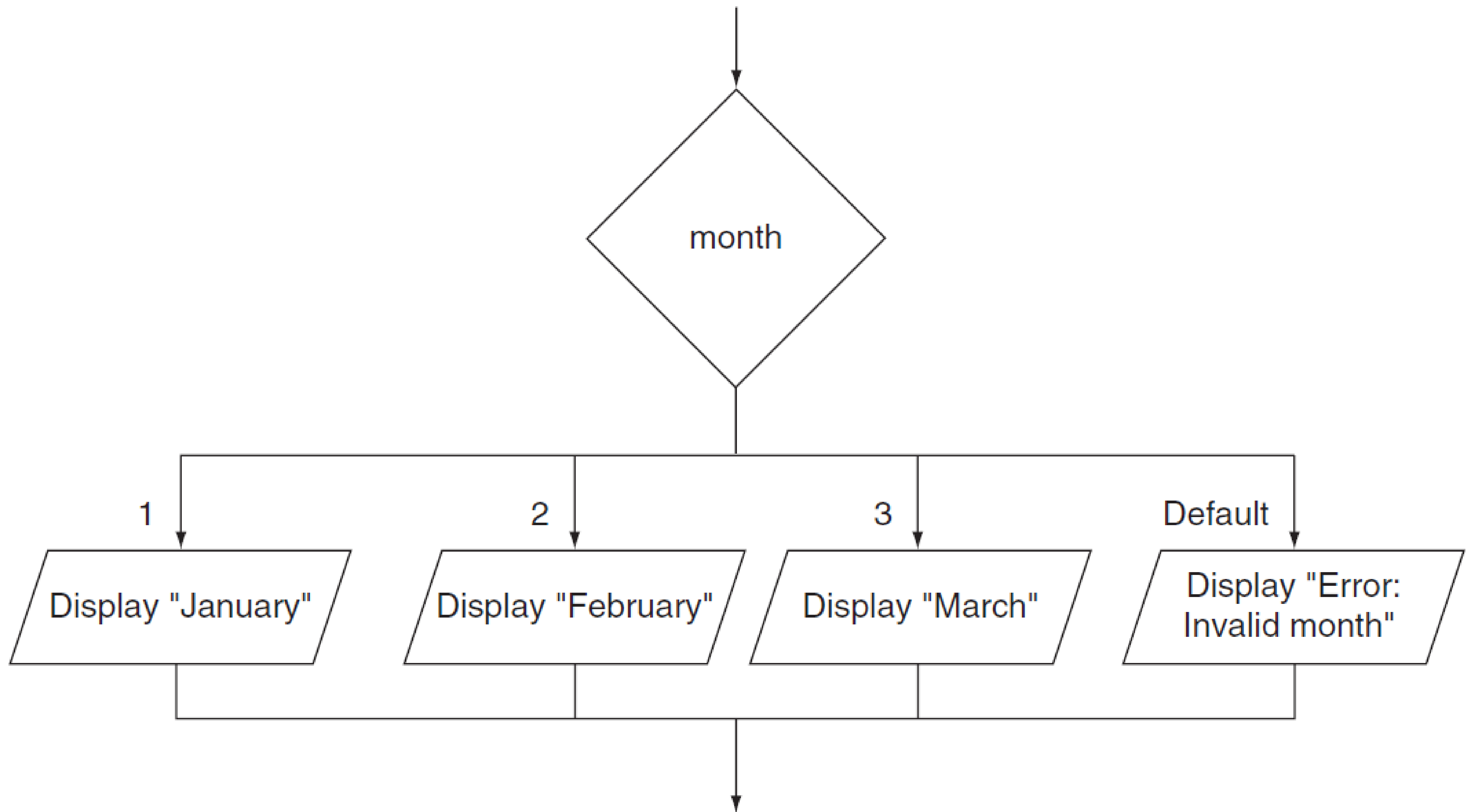
The `switch` Statement

- 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
    //as many times as necessary
  default:
    // place one or more statements here
}
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT


month
1
2
3
Default
Display "January"
Display "February"
Display "March"
Display "Error:
Invalid month"

The `switch` Statement

- The `switch` statement takes a value (`byte`, `short`, `int`, `long`, `char`, `string`) as the *SwitchExpression*.

```
switch (SwitchExpression)
{
   …
}
```

- The `switch` statement will evaluate the expression.
- If there is an associated `case` statement that matches that value, program execution will be transferred to that `case` statement.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `switch` Statement

- Each `case` statement will have a corresponding *CaseExpression* that must be unique.

```
case CaseExpression:
    // place one or more statements here
    break;
```

- If the *SwitchExpression* matches the *CaseExpression*, the Java statements between the colon and the `break` statement will be executed.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `case` Statement

- The `break` statement ends the `case` statement.
- The `break` statement is optional.
- Without the `break` statements, the program would execute all of the lines from the matching `case` statement to the end of the block.
- The `default` section is optional and will be executed if no *CaseExpression* matches the *SwitchExpression*.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The case Statement

Program output:
Enter 1, 2 or 3: *2 [Enter]*
You entered 2.

```
public static void main(String[] args)
{
        int number;

        Scanner keyboard = new Scanner(System.in);

        System.out.print("Enter 1, 2 or 3: ");
        number = keyboard.nextInt();

        switch (number)
        {
                case 1:
                        System.out.println("You entered 1.");
                        break;
                case 2:
                        System.out.println("You entered 2.");
                        break;
                case 3:
                        System.out.println("You entered 3.");
                        break;
                default:
                        System.out.println("That's not 1, 2 or 3!");
                        break;
        }
}
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `case` Statement

Program output:

Enter 1, 2 or 3: *1 [Enter]*
You entered 1.
You entered 2.
You entered 3.
That's not 1, 2 or 3!

```
public static void main(String[] args)
{
        int number;

        Scanner keyboard = new Scanner(System.in);

        System.out.print("Enter 1, 2 or 3: ");
        number = keyboard.nextInt();

        switch (number)
        {
                case 1:
                        System.out.println("You entered 1.");
                case 2:
                        System.out.println("You entered 2.");
                case 3:
                        System.out.println("You entered 3.");
                default:
                        System.out.println("That's not 1, 2 or 3!");
                break;
        }
}
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

```
public static void main(String[] args)
{
        String input;
        char foodGrade;

        Scanner keyboard = new Scanner(System.in);

        System.out.println("Our pet food is available in three grades:");
        System.out.print("A, B and C.  Which do you want pricing for? ");

        input = keyboard.nextLine();
        foodGrade = input.charAt(0);

        switch (foodGrade)
        {
                case 'a':
                case 'A':
                        System.out.println("30 cents per gram");
                        break;
                case 'b':
                case 'B':
                        System.out.println("20 cents per gram");
                        break;
                case 'c':
                case 'C':
                        System.out.println("10 cents per gram");
                        break;
                default:
                        System.out.println("Invalid choice");
                        break;
        }
}
```

Asks the user to select a grade of pet food

Stores the input as a character in variable *foodGrade*

No break statement

No break statement

No break statement

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

```
public static void main(String[] args)
{
		String input;
		char foodGrade;

		Scanner keyboard = new Scanner(System.in);

		System.out.println("Our pet food is available in three grades:");
		System.out.print("A, B and C.  Which do you want pricing for? ");

		input = keyboard.nextLine();
		foodGrade = input.charAt(0);

		switch (foodGrade)
		{
			case 'a':
			case 'A':
				System.out.println("30 cents per gram");
				break;
			case 'b':
			case 'B':
				System.out.println("20 cents per gram");
				break;
			case 'c':
			case 'C':
				System.out.println("10 cents per gram");
				break;
			default:
				System.out.println("Invalid choice");
				break;
		}
}
```

Program output:

Our pet food is available in three grades:
A, B and C. Which do you want pricing for? ***a [Enter]***
30 cents per gram

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

```
public static void main(String[] args)
{
        String input;
        char foodGrade;

        Scanner keyboard = new Scanner(System.in);

        System.out.println("Our pet food is available in three grades:");
        System.out.print("A, B and C.  Which do you want pricing for? ");

        input = keyboard.nextLine();
        foodGrade = input.charAt(0);

        switch (foodGrade)
        {
                case 'a':
                case 'A':
                        System.out.println("30 cents per gram");
                        break;
                case 'b':
                case 'B':
                        System.out.println("20 cents per gram");
                        break;
                case 'c':
                case 'C':
                        System.out.println("10 cents per gram");
                        break;
                default:
                        System.out.println("Invalid choice");
                        break;
        }
}
```

Program output:

Our pet food is available in three grades:
A, B and C. Which do you want pricing for? ***B [Enter]***
20 cents per gram

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Java Fundamentals

Output using `printf`

The `printf` Method

- You can use the `System.out.printf` method to perform formatted console output.
- The general format of the method is:

```
System.out.printf(FormatString, ArgList);
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

```
int hours = 40;
System.out.printf("I worked %d hours this week\n", hours);
```

Program output:

I worked 40 hours this week

When this string is printed, the %d will not be displayed. The value of the hours argument will be printed in the place of %d.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

```
System.out.printf(FormatString, ArgList);
```

`FormatString` is a string that contains text and/or special formatting specifiers.

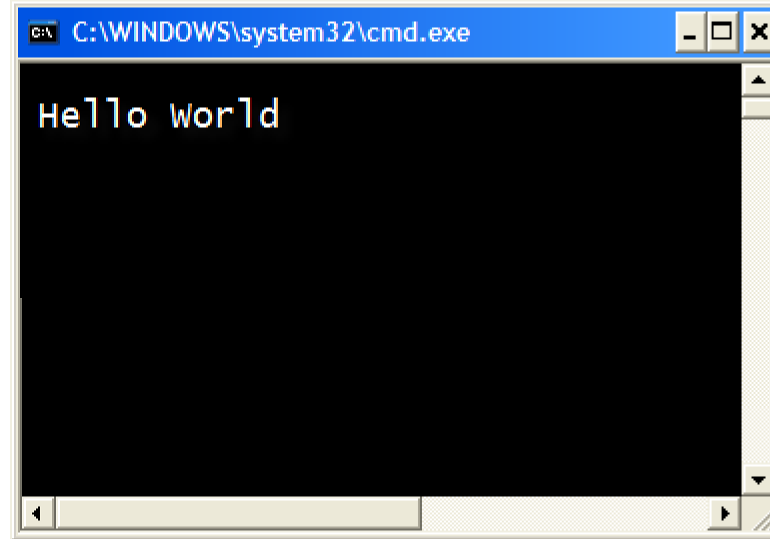
`ArgList` is optional. It is a list of additional arguments that will be formatted according to the format specifiers listed in the format string.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

- A simple example:

```
System.out.printf("Hello World\n");
```

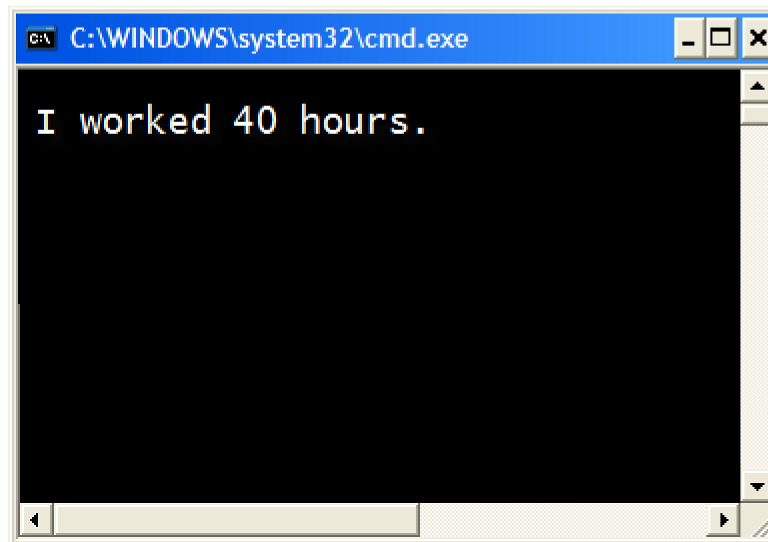


SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

- Another example:

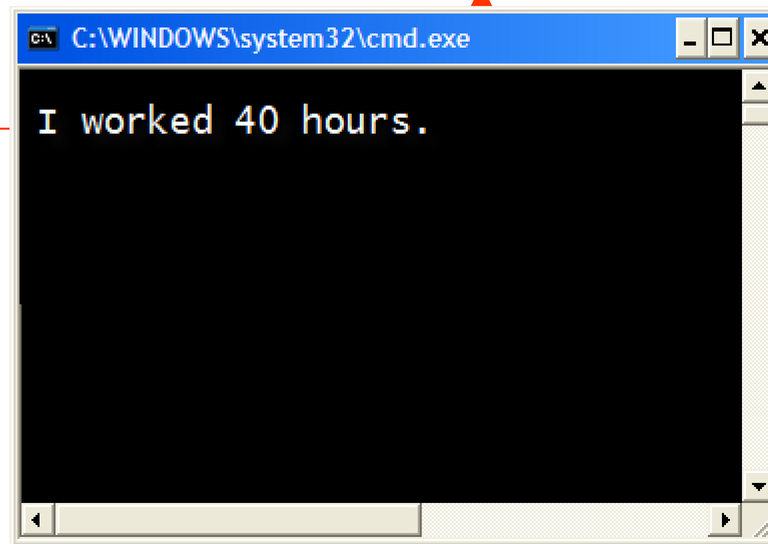
```
int hours = 40;
System.out.printf("I worked %d hours.\n", hours);
```



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

```
int hours = 40;
System.out.printf("I worked %d hours.\n", hours);
```



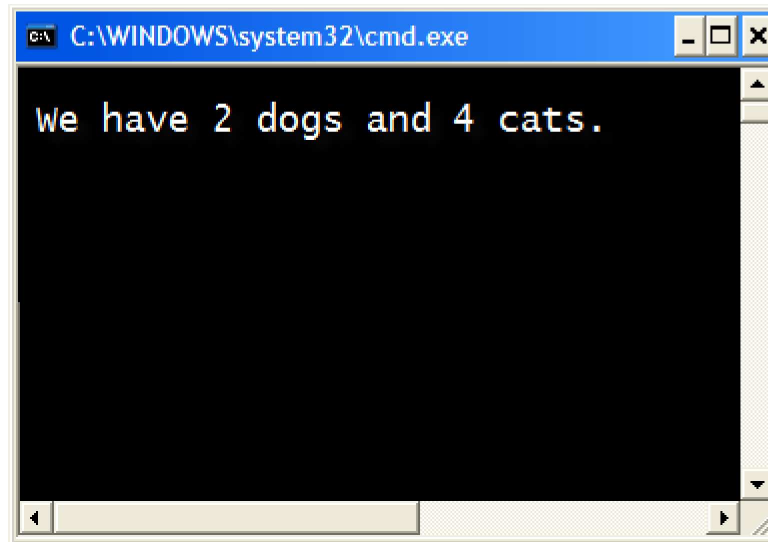
The `%d` format specifier indicates that a decimal integer will be printed.

The contents of the `hours` variable will be printed in the location of the `%d` format specifier.

The `printf` Method

- Another example:

```
int dogs = 2, cats = 4;
System.out.printf("We have %d dogs and %d cats.\n", dogs, cats);
```

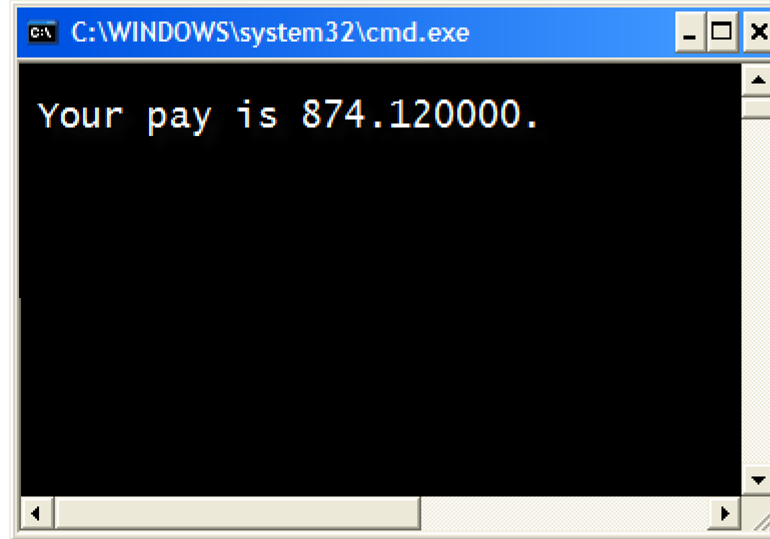


SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

- Another example:

```
double grossPay = 874.12;
System.out.printf("Your pay is %f.\n", grossPay);
```



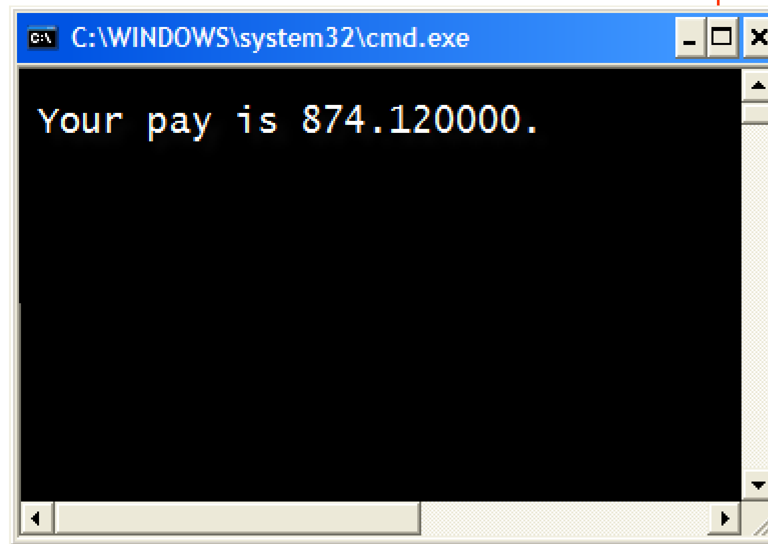
SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

- Another example:

```
double grossPay = 874.12;
System.out.printf("Your pay is %f.\n", grossPay);
```

The `%f` format specifier indicates that a floating-point value will be printed.

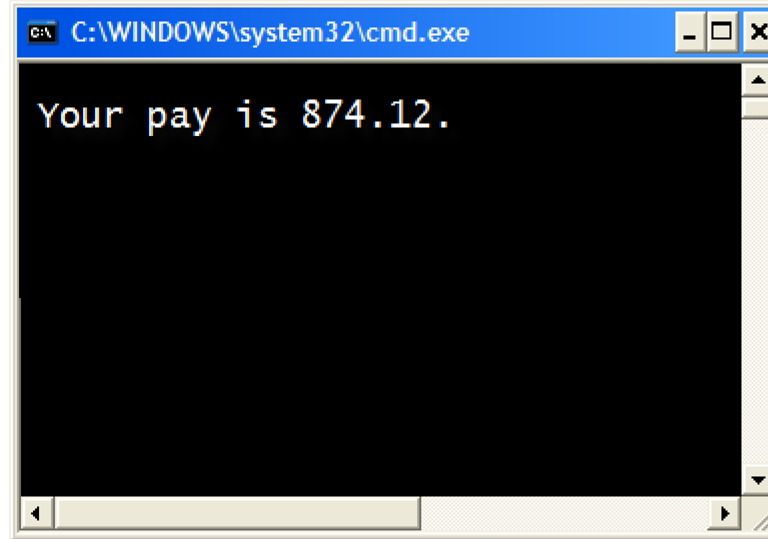


The contents of the `grossPay` variable will be printed in the location of the `%f` format specifier.

The `printf` Method

- Another example:

```
double grossPay = 874.12;
System.out.printf("Your pay is %.2f.\n", grossPay);
```

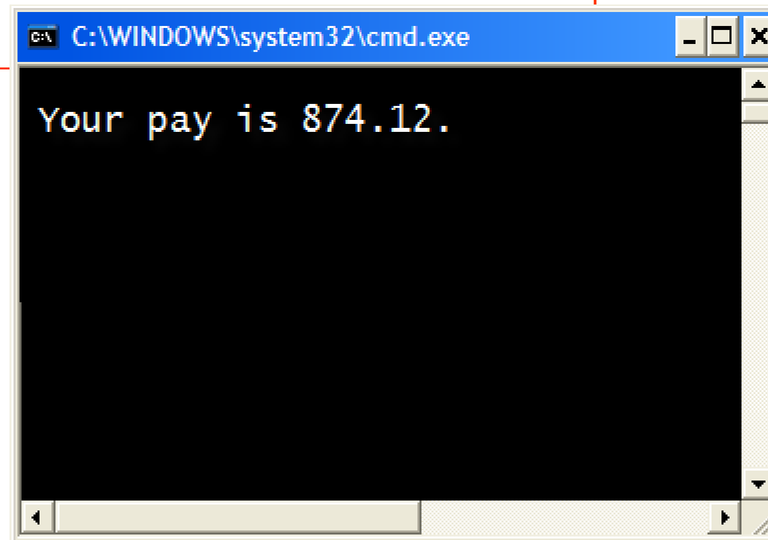


SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

- Another example:

```
double grossPay = 874.12;
System.out.printf("Your pay is %.2f.\n", grossPay);
```



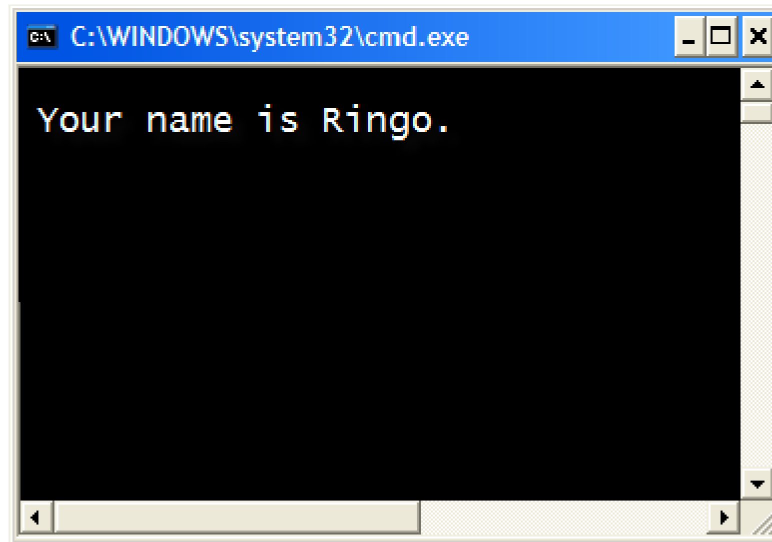
The `%.2f` format specifier indicates that a floating-point value will be printed, rounded to two decimal places.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `printf` Method

- Another example:

```
String name = "Ringo";
System.out.printf("Your name is %s.\n", name);
```



The %s format specifier indicates that a string will be printed.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Increment and Decrement

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Increment and Decrement Operators

- There are numerous times where a variable must simply be incremented or decremented.

```
number = number + 1;
number = number – 1;
```

- Java provide shortened ways to increment and decrement a variable’s value.
- Using the ++ or -- unary operators, this task can be completed quickly.

```
number++;  or  ++number;
number--;  or  --number;
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Differences Between Prefix and Postfix

- When an increment or decrement are the only operations in a statement, there is no difference between prefix and postfix notation.
- When used in an expression:
  - prefix notation indicates that the variable will be incremented or decremented prior to the rest of the equation being evaluated.
  - postfix notation indicates that the variable will be incremented or decremented after the rest of the equation has been evaluated.

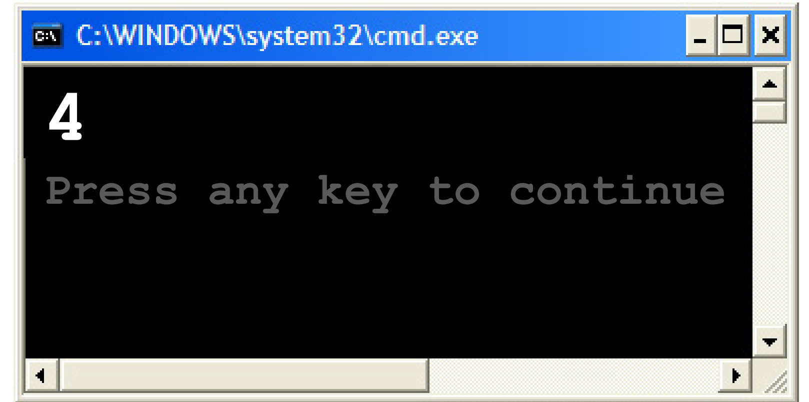
SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Differences Between Prefix and Postfix

```
int number = 4;
System.out.println(number++);
```

Postfix

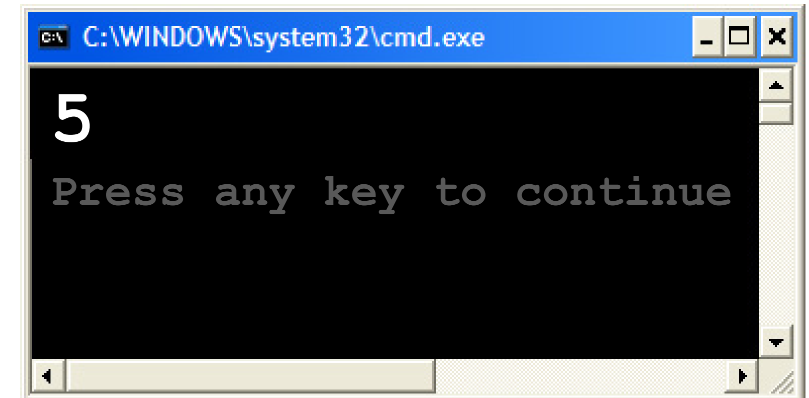
number is **displayed** on the screen first, and then **incremented**.




```
int number = 4;
System.out.println(++number);
```

Prefix

number is **incremented** first, and then **displayed** on the screen



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Differences Between Prefix and Postfix

```
int x = 1, y;
y = x++;
```

Postfix

x is defined as an int and initialised with the value 1. **y** is declared an int.

The value of **x** is assigned to the variable **y** and then **incremented**

At the end of this statement: x = 2, y = 1

```
int x = 1, y;
y = ++x;
```

Prefix

x is defined as an int and initialised with the value 1. **y** is declared an int.

The value of **x** is **incremented** and then assigned to variable **y**

At the end of this statement: x = 2, y = 2

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

While loops

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `while` Loop

- Java provides three different looping structures.
- The `while` loop has the form:

```
while(condition)
{
  statements;
}
```

- While the condition is true, the statements will execute repeatedly.
- The `while` loop is a *pretest* loop, which means that it will test the value of the condition prior to executing the loop.

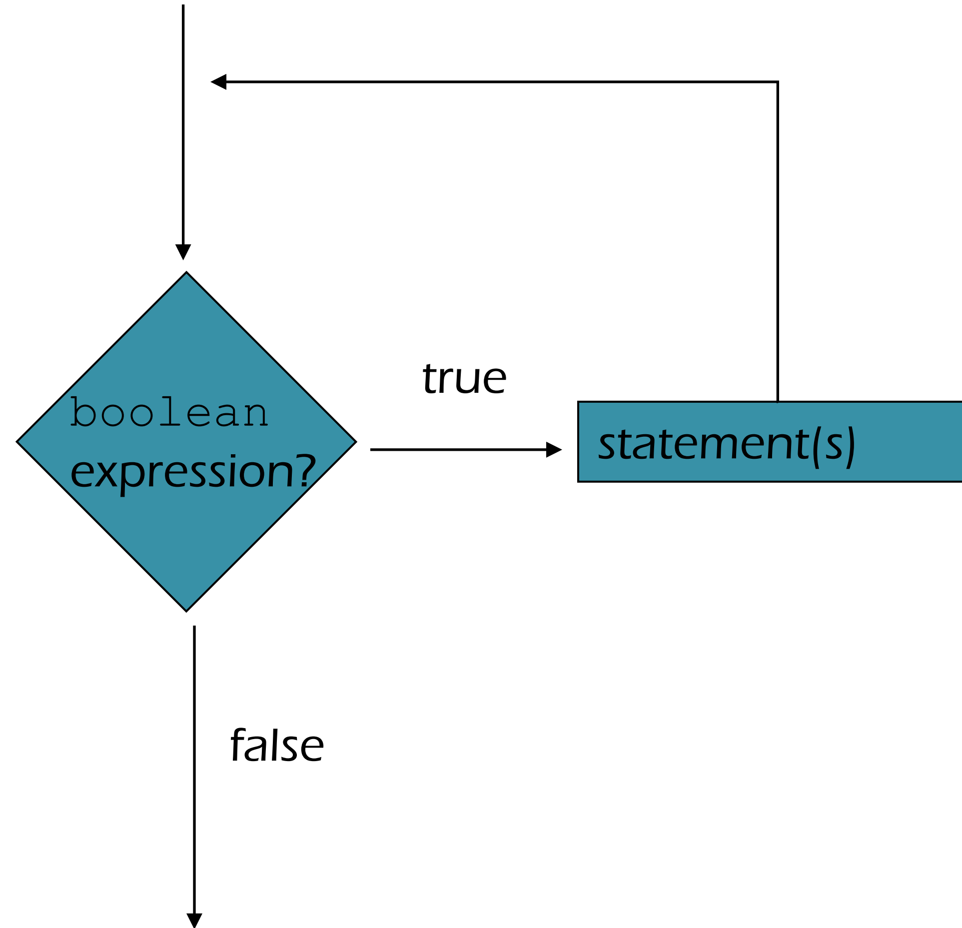
SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `while` Loop

- Care must be taken to set the condition to false somewhere in the loop so the loop will end.
- Loops that do not end are called *infinite loops*.
- A `while` loop executes 0 or more times. If the condition is false, the loop will not execute.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `while` Loop Flowchart



The `while` Loop

```
public class WhileLoop
{
	public static void main(String[] args)
	{
		int number = 1;

		while (number <=5)
		{
			System.out.println("Hello");
			number++;
		}

		System.out.println("That's all!");
	}
}
```

number declared and initialised with the value of *1*

Tests variable *number* to determine whether it is less than or equal to *5*

If it is, then these statements are executed.

When *number <= 5* is tested and found to be false, the loop will terminate and the program will resume execution at the statement that immediately follows the loop.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `while` Loop

6

number <=5

True

Print "Hello"

Add 1 to number

False

Print "That's all"

Program output
Hello
Hello
Hello
Hello
Hello
That's all!

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Infinite Loops

- In order for a `while` loop to end, the condition must become false. The following loop will not end:

```
int x = 20;
while(x > 0)
{
  System.out.println("x is greater than 0");
}
```

- The variable `x` never gets decremented so it will always be greater than 0.
- Adding the **`x--`** above fixes the problem.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Infinite Loops

- This version of the loop decrements `x` during each iteration:

```
int x = 20;
while(x > 0)
{
    System.out.println("x is greater than 0");
    x--;
}
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Block Statements in Loops

- Curly braces are required to enclose block statement while loops. (like block `if` statements)

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

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `while` Loop for Input Validation

- *Input validation* is the process of ensuring that user input is valid.

```
System.out.print("Enter a number in the range of 1 through 100: ");
number = keyboard.nextInt();

// Validate the input.
while (number < 1 || number > 100)
{
  System.out.println("That number is invalid.");
  System.out.print("Enter a number in the range of 1 through 100: ");

  number = keyboard.nextInt();
}
```

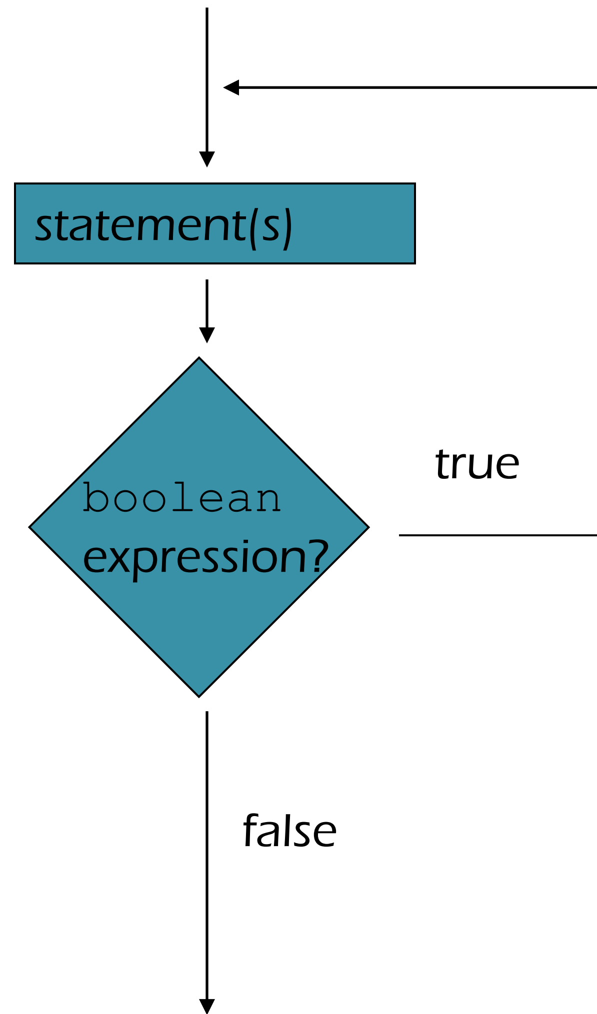
SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The `do-while` Loop

- The `do-while` loop is a *post-test* loop, which means it will execute the loop prior to testing the condition.
- The `do-while` loop (sometimes called a `do` loop) takes the form:

```
do
{
  statement(s);
}while (condition);
```

The `do-while` Loop Flowchart



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

```
public static void main(String[] args)
{
        int score1, score2;
        double average;
        char repeat;
        String input;

        Scanner keyboard = new Scanner(System.in);

        do
        {
                System.out.print("Enter score #1:  ");
                score1 = keyboard.nextInt();
                System.out.print("Enter score #2:  );
                score2 = keyboard.nextInt();
                keyboard.nextLine();

                average = (score1 + score2) / 2.0;
                System.out.println("The average is : " + average);

                System.out.println("Would you like to average another set of test scores?");
                System.out.print("Enter 'Y' for yes or 'N' for no: ");
                input = keyboard.nextLine();
                repeat = keyboard.charAt(0);

        }   while (repeat == 'Y' || repeat == 'y');
}
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The char Data Type

Unicode

Characters in Java are represented by numerical values using Unicode, which is an international standard that assigns a unique number to every character. Unicode covers almost all written languages, and in Java, it requires 2 bytes per character, allowing for the representation of over 65,000 characters.

Characters are internally represented by numbers. Each character is assigned a unique number.

Java uses Unicode, which is a set of numbers that are used as codes for representing characters. Each Unicode number requires two bytes of memory, so ***char*** variables occupy ***two bytes***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

```
1       public class Characters
2       {
3               public static void main(String[] args)
4               {
5                       char letter;
6
7                       letter = 65;
8                       System.out.println(letter);
9                       letter = 66;
10                      System.out.println(letter);
11              }
12      }
```

Printed on the computer screen when application runs:

```
A
B
```

Characters in Java are stored using Unicode values, making them ordinal. This means that characters can be compared using relational operators, such as <, >, <=, >=, ==, and !=.

For example, in the ASCII range, the character 'A' is less than 'Z' because 'A' has a smaller Unicode value than 'Z'. This makes comparing characters useful in sorting algorithms, checking ranges, and validating input.

- Characters can be tested with relational operators.
- Characters are stored in memory using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are *ordinal*, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

```
char c = 'A';
if(c < 'Z')
  System.out.println("A is less than Z");
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

In Java, percentages are represented as decimals. For instance, 50% is represented as 0.5. When calculating percentages, be mindful of this conversion to ensure accurate results.

For instance:
100% is represented as 1.0
50% is represented as 0.5
20% is represented as 0.2
5% is represented as 0.05

$$50\% = \frac{50}{100} = 0.5$$

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Math Class

In Java, the Math class provides basic mathematical functions and constants, like addition, square roots, trigonometry, and more.

It includes methods like Math.sqrt() for square roots, Math.pow() for powers, and constants like Math.PI for π.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Math Class

The *Math.pow* Method

The Math.pow method is used to raise a number to the power of another number. It takes two double arguments. The first argument is the base, and the second is the exponent.

Example Code:

```
double result = Math.pow(3.0, 2.0);  // 3^2 = 9
System.out.println("3 raised to the power of 2: " + result);
```

The *Math.pow* method raises a number to a power.

result = Math.pow(3.0, 2.0);

This method takes two ***double*** arguments. It raises the first argument to the power of the second argument, and returns the result as a ***double***.

$$result = 3^2$$

$$result = 9.0$$

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Math Class

The *Math.sqrt* Method

The Math.sqrt method returns the square root of a number. It takes a double value as an argument and returns the square root of the number.

Example Code:

```
double result = Math.sqrt(4.0);  // Square root of 4 is 2
System.out.println("Square root of 4: " + result);
```

The *Math.sqrt* method accepts a double value as its argument and returns the square root of the value.

```
result = Math.sqrt(4.0);
```

$$result = \sqrt{4}$$

$$result = 2.0$$

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The Math Class

The *Math.PI* predefined constant

The *Math.PI* constant is a constant assigned with a value of 3.14159265358979323846, which is an approximation of the mathematical value pi.

```
double area, radius;

radius = 5.5;
area = Math.PI * radius * radius;
```

$$area = \pi \times 5.5 \times 5.5$$

$$area = 95.03317777$$

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Combined Assignment Operators

Combined assignment operators in Java are shorthand operators that perform an operation on a variable and then assign the result back to that variable in a single step. They make code shorter and often easier to read.

Combined Assignment Operators

```
x = x +1;
```

Faster way:

```
x += 1;
```

On the right of the assignment operator, 1 is added to x. The result is then assigned to x, replacing the previous value. Effectively, this statement adds 1 to x.

```
y = y -1;
```

Faster way:

```
y -= 1;
```

On the right of the assignment operator, 1 is subtracted from y. The result is then assigned to y, replacing the previous value. Effectively, this statement subtracts 1 from y.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Combined Assignment Operators

```
z = z* 10;
```

Faster way:

```
z *= 10;
```

On the right of the assignment operator, 10 is multiplied by z. The result is then assigned to z, replacing the previous value. Effectively, this statement multiplies z with 10.

```
a = a / b;
```

Faster way:

```
a /= b;
```

On the right of the assignment operator, a is divided by b. The result is then assigned to a, replacing the previous value. Effectively, this statement divides a by b.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Combined Assignment Operators

x = x % 4;

Faster way:

x %= 4;

On the right of the assignment operator, the remainder of x divided by 4 is calculated. The result is then assigned to x, replacing the previous value. Effectively, this statement assigns the remainder of x/4 to x.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Sentinel Values

A sentinel value is a special value used to terminate loops or signal the end of data processing. It is often used when the number of inputs or records is unknown, and a special value is chosen to signal the end.

This technique is especially useful in file processing and user input where the total data length isn't predetermined.

For instance, a sentinel value might be a negative number when processing positive integers, or a specific string like 'END'. When the program encounters this value, it knows to stop further data collection or processing.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types:

Casting

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types

Before a value can be stored in a variable, the ***value's data type*** must be compatible with the ***variable's data type***. Java performs some conversions between data types automatically, but does not automatically perform any conversion that can result in the loss of data.

Conversion between Primitive Data Types

```
int x;
double y = 2.8;
x = y;
```

This statement is attempting to store a ***double*** value (2.8) in an ***int*** variable. This will give an error message. A ***double*** can store fractional numbers and can hold values much larger than an ***int*** can hold. If this were permitted, a loss of data would be likely.

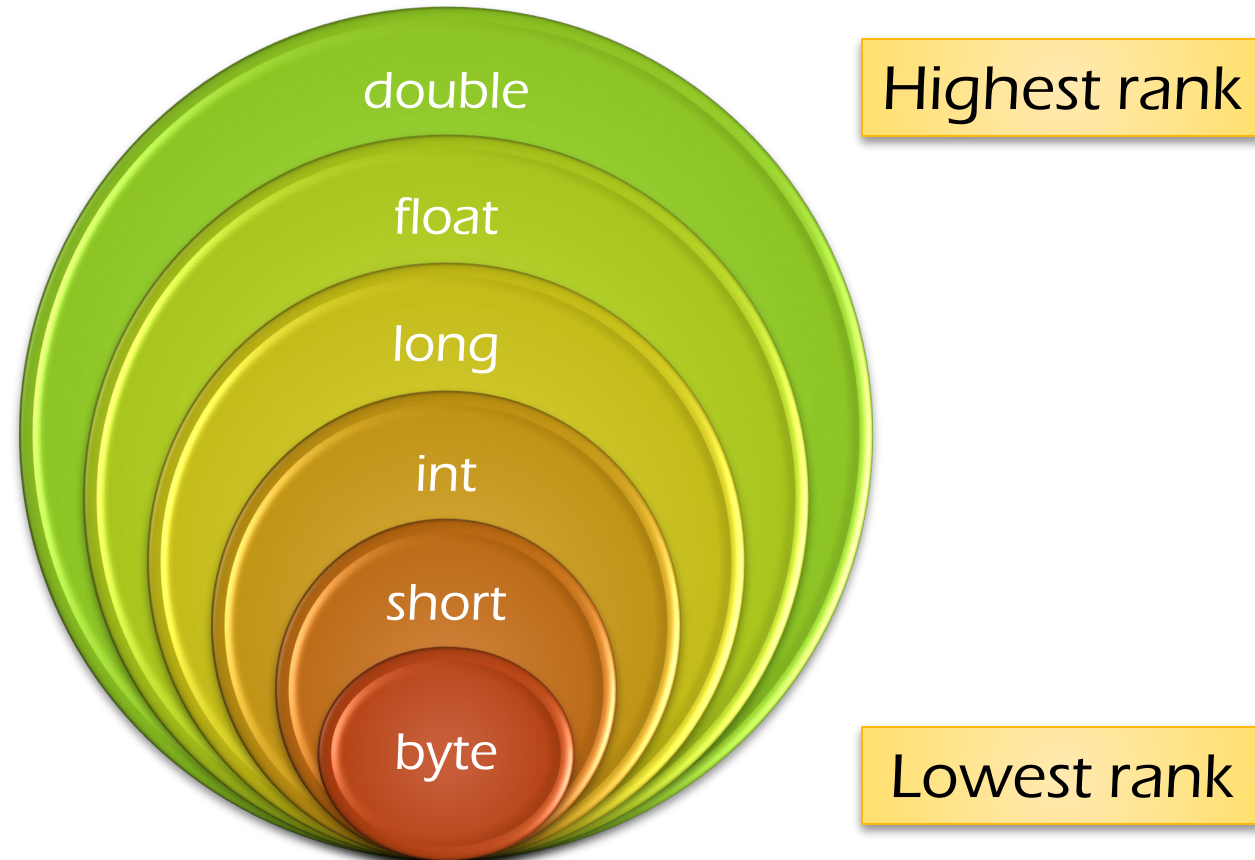
SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types

```
int x;
short y = 2;
x = y;
```

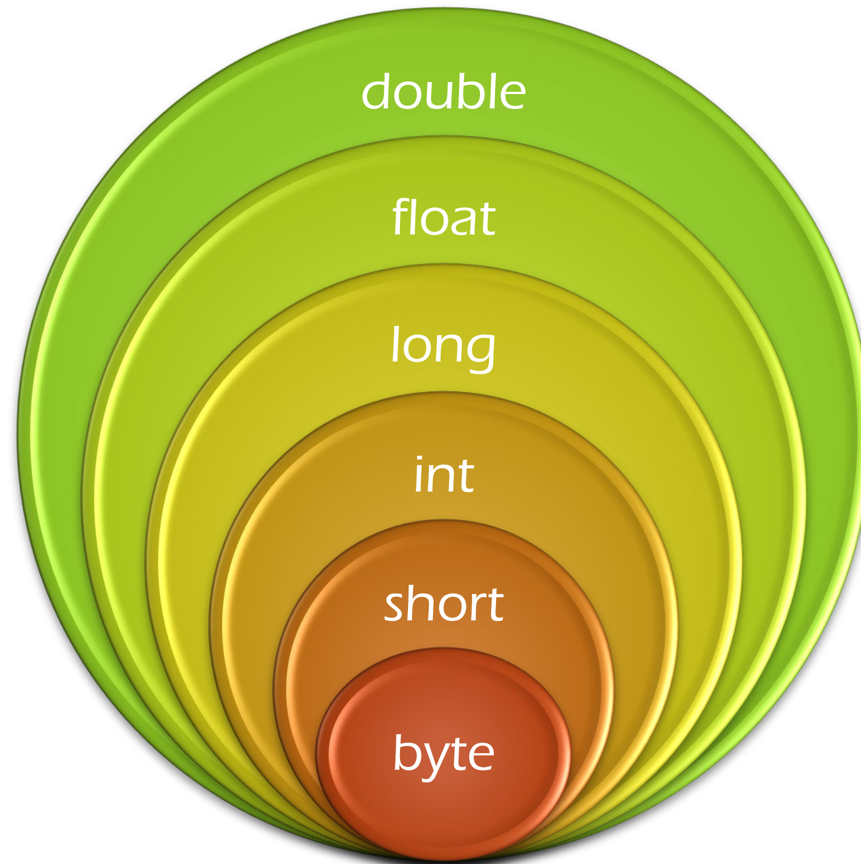
This statement is attempting to store a *short* value (2) in an *int* variable. **This will work with no problems.**

Conversion between Primitive Data Types



SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types



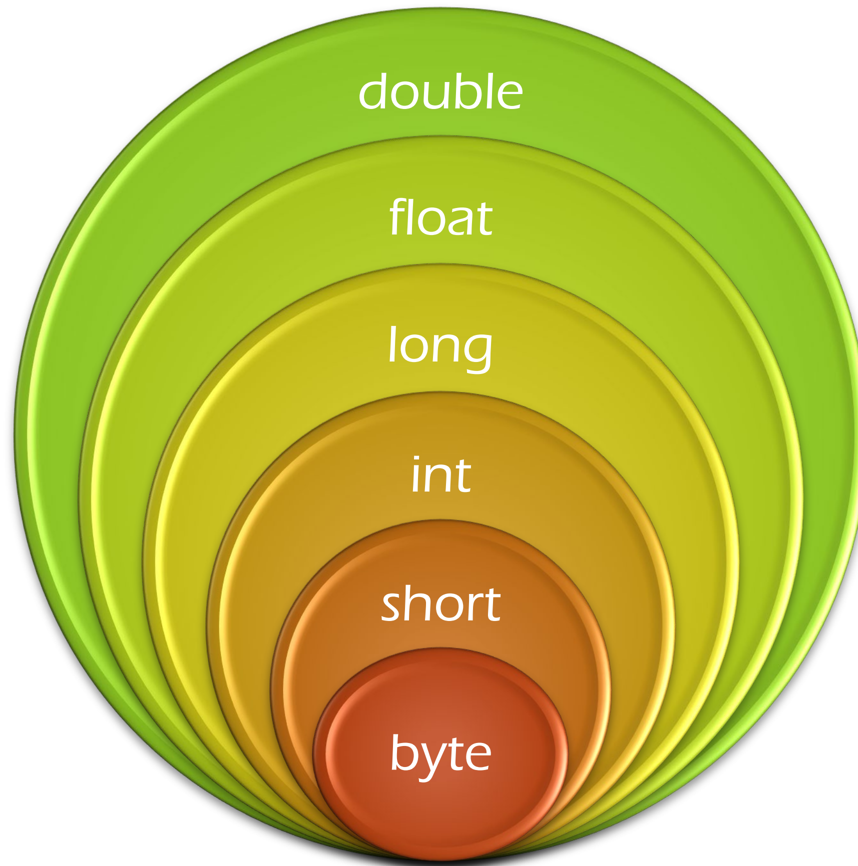
In assignment statements where values of ***lower-ranked data types*** are stored in variables of ***higher-ranked data types***, Java automatically converts the lower-ranked value to the higher-ranked type.

```
double x;
int y = 2;
x = y;
```

We call this a widening conversion

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types



In assignment statements where values of ***higher-ranked data types*** are stored in variables of ***lower-ranked data types***, Java does not automatically perform the conversion because of possible data loss.

```
int x;
double y = 2.0;
x = y;
```

Error!

We call this a narrowing conversion

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types

(Cast operators)

The cast operator lets you manually convert a value,
even if it means that a *narrowing conversion* will take place.

```
int x;
double y = 2.5;
x = y;
```

Error!

Cast operator

```
int x;
double y = 2.5;
x = (int) y;
```

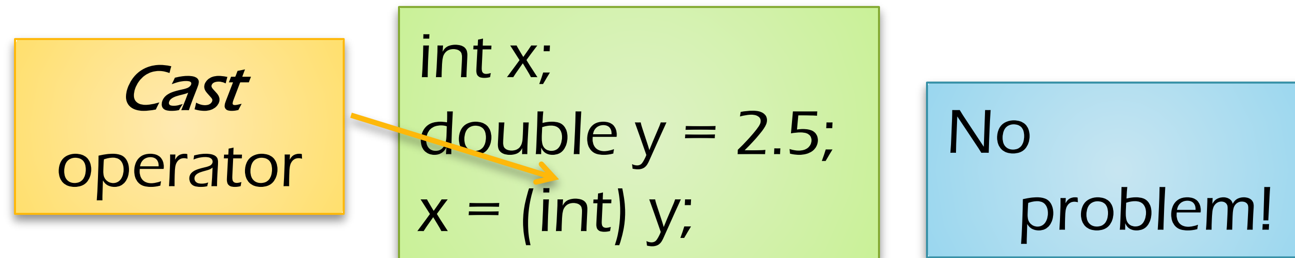
No problem!

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Conversion between Primitive Data Types

(Cast operators)

Java compiles the code with the cast operator with no problems. In this case variable y has a value of 2.5 (floating-point value) which must be converted to an integer.



The value that is returned and stored in variable ***x*** would be truncated, which means the fractional part of the number is lost to accommodate the ***integer*** data type.

Thus: ***x*** = 2

The value of variable y is not changed at all: ***y*** = 2.5

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Mixed Integer Operations

One of the nuances of the Java language is the way it handles arithmetic operations on *int*, *byte* and *short*.

When values of the *byte* or *short* data types are used in arithmetic expressions, they are temporarily converted to *int* values.

```
short x = 10, y = 20, z;
z = x + y;
```

Error!

How can this error be rectified?

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Mixed Integer Operations

```
short x = 10, y = 20, z;
z = x + y;
```

Error!

The error results from the fact that *z* is a short. The expression *x + y* results in an *int* value.

This can be corrected if *z* is declared as an *int*, or if a cast operator is used.

```
short x = 10, y = 20;
int z;
z = x + y;
```

No problem!

```
short x = 10, y = 20, z;
z = (short) (x + y);
```

No problem!

Creating Named Constants with ***final***

The final key word can be used in a variable declaration to make the variable a named constant. Named constants are initialized with a value, and that value cannot change during the execution of the program.

```
amount = balance * 0.072;
```

The 1st problem that arises is that it is not clear to anyone but the original programmer as to what the 0.072 is.

The 2nd problem occurs if this number is used in other calculations throughout the program and must be changed periodically.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Creating Named Constants with *final*

We can change the code to use the *final* key word to create a constant value.

```
amount = balance * 0.072;
```

Old code

```
final double INTEREST_RATE = 0.072;

amount = balance * INTEREST_RATE;
```

New code

Now anyone who reads the code will understand it. When we want to change the interest rate, we change it only once at the declaration.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The *String* Class

String Methods

Because the ***String*** type is a ***class*** instead of a ***primitive data type***, it provides numerous ***methods*** for working with strings.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The *String* Class

charAt() Method

This method returns the ***character*** at the specified ***position***.

```
char letter;
String name = “Arnold”;
letter = name.charAt(2);
```

0	1	2	3	4	5
A	*r*	*n*	*o*	*l*	*d*

After this code executes, the variable ***letter*** will hold the character ***‘n’***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The *String* Class
length() Method

This method returns the *number of characters* in a string.

```
int stringSize;
String name = "Arnold";
stringSize =
     name.length();
```

0	1	2	3	4	5
A	*r*	*n*	*o*	*l*	*d*

After this code executes, the variable *stringSize* will hold the value *6*.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The ***String*** Class
toLowerCase() Method

This method returns a new string that is the *lowercase* equivalent of the string contained in the calling object.

```
String bigName = “ARNOLD”;
String littleName = bigName.toLowerCase();
```

After this code executes, the variable *littleName* will hold the string *“arnold”.*

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The *String* Class

toUpperCase() Method

This method returns a new string that is the *uppercase* equivalent of the string contained in the calling object.

```
String littleName = "arnold";
String bigName = littleName.toUpperCase();
```

After this code executes, the variable *bigName* will hold the string *"ARNOLD".*

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The char Data Type

Unicode

Characters are internally represented by numbers. Each character is assigned a unique number.

Java uses Unicode, which is a set of numbers that are used as codes for representing characters. Each Unicode number requires two bytes of memory, so ***char*** variables occupy ***two bytes***.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

The char Data Type

```
1	public class Characters
2	{
3		public static void main(String[] args)
4		{
5			char letter;
6
7			letter = 65;
8			System.out.println(letter);
9			letter = 66;
10			System.out.println(letter);
11		}
12	}
```

Printed on the computer screen when application runs:

A

B

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Comparing Characters

- Characters can be tested with relational operators.
- Characters are stored in memory using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are *ordinal*, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

```
char c = 'A';
if(c < 'Z')
  System.out.println("A is less than Z");
```

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Importance of Coding Styles

Coding styles play a crucial role in software development:

1. Improves readability, making it easier for others to understand the code.

2. Promotes consistency across a codebase, especially in team projects.

3. Makes debugging and maintaining code simpler and more efficient.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Different Coding Styles

Different coding styles serve different purposes and preferences:

1. Some styles prioritize compactness (e.g., K&R), while others focus on clarity (e.g., Allman).

2. The key is consistency within the project or organization.

3. Adapting to a style is important:

> *Adapting to a coding style within a project is essential for readability and maintainability. Consistent styling helps anyone working on the code understand its structure more easily, making it straightforward to read, debug, and modify. When code follows a predictable style, it reduces the cognitive load, allowing developers to focus on the logic rather than deciphering varied formatting.*

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT

Why Indentation Matters

Indentation is crucial for enhancing the structure and readability of code:

1. It visually separates code blocks, making the control flow clearer.

2. Proper indentation reduces errors by clearly showing the hierarchy of logic.

3. Inconsistent indentation can lead to confusion, even in small projects.

SOFTWARE DESIGN
& PROGRAM DEVELOPMENT