COMP-202
Unit 2: Programming Basics

CONTENTS:
The Java Programming Language
Variables and Types
Basic Input / Output
Expressions
Conversions
Part 1: The Java Programming Language
The Java Programming Language

- Java was created by Sun Microsystems, Inc.
- It was introduced in 1995 and has become quite popular
- It is an *object-oriented* language
  - It represents the program as a series of *objects*, each of which belongs to one of many categories called *classes*
- The Java programming language combines both compilation and interpretation
Java Translation and Execution (1)

• The Java compiler translates Java source code into a special representation called *bytecode*
• Java bytecode is not the machine language for any traditional CPU
• Another software tool, an interpreter called the *Java virtual machine* (JVM) translates bytecode into machine language and executes it
• Therefore the Java compiler is not tied to any particular machine
  – Java bytecode will run on any CPU for which there exists a JVM
• Java is considered to be *architecture-neutral*
Java Translation and Execution (2)

This is the standard approach to Java translation and execution.
This approach to Java translation and execution is very uncommon, and probably unnecessary.
Java Program Structure

• In the Java programming language:
  – A program consists of one or more classes; each class is defined in a different file
  – A class contains one or more methods
  – A method contains program statements
  – Statements are the actual commands you issue, the instructions that are executed when the program runs

• A Java program always has at least one class which contains a method called main()
  – This is where the program starts executing

• For now, all programs we develop will involve writing only one class, and this class will contain exactly one method: main()
Java Program Structure: Classes

public class MyProgram  
{ 

class header: the name of the class

The class header states that you are defining a new class whose name will be the name you specify. This class will consist of what you write in the class body.

Important note: the name of the file which contains this class MUST be the name of the class, followed by extension .java

Here: MyProgram.java
Java Classes

• There are three possible roles for a class in Java
  – A class can be the starting point of a program; any class that defines a `main()` method has this role
  – A class can be a collection of related methods
  – A class can define a category of objects, and specify what objects that belong to this category look like and how they behave

• Many classes only have one of these roles, but some have two and even all three

• For now, all the classes we write will have only one role: they will be the starting point of a program
Java Program Structure: Methods

```java
public class MyProgram {

    public static void main(String[] args) {

        method header

        method body

        The method header states that you are defining a new method; this method consists of the statements in the method body. The method header also specifies the new method's name.

    }

    }

```
Java Methods

• A Java method is a list of statements that has been given a name
  – The statements that the method consists of perform a task when they are executed together

• *Writing a method* means specifying a list of statements and giving a name to this list of statements

• Once a method is written, its name can be used as a single statement inside another method
  – When that single statement is executed, all statements in the method are executed
  – This is referred to as *calling* or *invoking* a method

• We will cover methods in detail later
Part 2: Java Basics
Our First Java Program (1)

• Here is our first Java program:

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

• Some highlights:
  – It involved writing a single class called `HelloWorld`
  – This class contains a single method called `main()`, which is where the program starts
  – The `main()` method contains a single statement, which requests that the sentence "Hello world!" (without the quotation marks) be displayed on the screen
Our First Java Program (2)

• By looking at the program and knowing what it does, can you deduce a way to modify it so that it does the following:
  – Display "Programming is fun!" instead of "Hello world!"?
  – Display two different sentences?
import java.util.Scanner;

/* This program asks the user to enter an integer, reads this integer from the keyboard, and displays it to the screen. */
public class EchoNumber {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int number;

        // Prompt the user for a number, read it, and display it.
        System.out.print("Enter a whole number: ");
        number = keyboard.nextInt();
        System.out.println("You entered the following number "+
                         number);
    }
}
Dissecting EchoNumber.java

- Although very simple, the program in `EchoNumber.java` illustrates a number of basic concepts of Java programming:
  - Comments
  - White space and formatting
  - Identifiers
  - Variables
  - Data types
  - Assignment statements
  - Reading information from the keyboard
  - Displaying information to the screen

- We will now cover each of these concepts in detail one by one
Comments (1)

- Comments are notes in a program
  - They are meant for *humans*, and are completely ignored by the compiler
  - You can write anything in a comment
  - In most cases, they describe aspects of the program such as its purpose or the way it works

- Syntax
  - Comments that start with `//` end when the current line ends; the compiler ignores everything between `//` and the end of the current line
  - Comments that start with `/*` end with `*/` ; the compiler ignores everything between `/*` and the matching `*/`
Comments (2)

• Comment examples:

```c
// This is a one line comment.
/* This is a multi-line comment. */
```

• Even though they do not change how the program works, comments are important!
  – They provide valuable information to humans who read the program about its purpose and the way it works, including to the one who wrote it in the first place!
  – "You know you're brilliant, but maybe you'd like to understand what you did 2 weeks from now." - Linus Torvalds, Linux 1.3.53

CodingStyle documentation
Formatting Rules

• Spaces, blank lines, and tabs are collectively called *white space*
  – It is used to separate words and symbols in a program
  – Extra white spaces are completely ignored

• A valid Java program can be formatted many different ways

• Programs should be formatted for *readability by humans*
  – Use proper indentation
  – Use space and new lines
  – Use comments

• The program in *EchoNumber.java* is formatted very nicely
Bad Formatting (1)

• The following is an example of a (very) badly formatted program:

```java
import java.util.Scanner;
public class EchoNumberBad { public static void main(String[] args){ Scanner keyboard = new Scanner(System.in); int number;
System.out.print("Enter a whole number: ");
number = keyboard.nextInt();
System.out.println("You entered the following number "+ number); }}
```
Bad Formatting (2)

• As long as a program is syntactically correct, the compiler will understand it, no matter how badly formatted it is
• However, humans who have to read your program (including TAs) may have trouble understanding it if it is badly formatted
• Make sure to format your programs so that they are readable by humans
Identifiers (1)

• *Identifiers* are the words a programmer uses in a program
• They are used to give names to things
• Examples of things that can be named with identifiers:
  – Classes
  – Methods
  – Variables
  – Constants
• An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign ($)
• Identifiers cannot begin with a digit
• Java is *case sensitive*, therefore `Result` and `result` are different identifiers
Identifiers (2)

• Sometimes we choose identifiers ourselves when writing a program (such as number, keyboard, or EchoNumber)
  – We should choose identifiers that are meaningful so that the program is more readable for humans

• Sometimes we are using another programmer's code, so we use the identifiers that they chose (such as println())

• Often we use special identifiers called reserved words that already have a predefined meaning in the language
  – A reserved word cannot be used in any other way; it cannot be used to give names to things like other identifiers
  – Examples of reserved words: class, public, static, void
  – There are many others
Java Reserved Words

• Here is the complete list of Java reserved words:

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

do  instanceof  static

strictfp
Identifiers: Exercise

- Which of the following identifiers are invalid according to the Java rules for identifiers? Why are they invalid?
  - myIdentifier
  - _my_other_identifier
  - yet-another-identifier
  - $can
  - 2for1
  - twoFor1
  - class
  - my_class
Identifier Conventions

• There exist Java *style* conventions for identifiers
  – Do not put characters between words
  – The first letter of a word should be in upper-case
  – The first letter of the first word should be in lower-case unless the identifier is for a class
  – The first word of the identifier for a method should be a verb
  – Identifiers should be descriptive and avoid abbreviations

• The compiler will not complain if you do not follow the conventions, but not following conventions considered bad practice

• See the *General Instructions and Regulations for Assignments* for further information regarding conventions for identifiers
Conventions: Exercise

• Which identifiers follow style conventions for Java?
  – myIdentifier
  – _my_other_identifier
  – totalValue
  – totVal
  – theSumOfAllTheValuesEnteredByTheUser
  – MyClass
  – my_class
  – getValue
  – GetValue
Variables (1)

• A variable is a placeholder for values
  – We can store a value in a variable
  – We can use the value stored in a variable to compute other values (which may be stored in other variables), to make decisions, to display it, or for other things
  – Before we use the value that a variable contains, we must store a value in this variable, otherwise the compiler will report an error

• Each variable has a name
  – The compiler allocates one or more consecutive memory cells
  – The (combined) contents of these cells forms the value of the variable
  – The compiler assigns the name chosen for the variable to these cells
Variables (2)

- We can then refer to the value stored in these memory cells using the variable name
- Much easier, more practical, and more flexible than using the addresses of the memory cells (and less error-prone)

• Each variable also has a type
  - The type of a variable specifies the kind of values it can contain
Variable Declarations

• Before we can store a value in a variable or use the value it contains, we have to declare it.

• A variable declaration is a statement that announces that we want to create and use a new variable.
  – They can occur anywhere in a method, but they are typically at the beginning.
  – They must indicate the name and the type of the new variable.

• Once a variable has been declared, we can store a value in it or use the value it contains.

• If you try to store a value in a variable before it is declared, or use the value stored in a variable before the variable is declared, the compiler will generate an error.
Syntax for Declaring Variables

```plaintext
modifiers  type  identifier  =  value;
```

Where:

- `modifiers` final, ...
- `type` int, double, char
- `identifier` as defined previously
- `= value` an expression matching the type
- `;` (optional) (mandatory) (mandatory) (optional) (mandatory)

This is a partial definition
More on Variable Declarations

• Note that you can declare more than one variable with one variable declaration statement
  – The variable names are separated using commas
• For example, the following statement declares three different variables, called var1, var2, and var3, each of type int:
  
  ```
  int var1, var2, var3;
  ```
Types

• A *type* is a category of values that a variable belongs to and determines:
  – How to interpret the value it contains
  – What are the possible values it can contain
• For example, it makes no sense to assign the name of a month as a value to a variable whose purpose is to hold the name of a day of the week
• In Java, all variables and all values have a type
The Purpose of Types

• The type of a variable provides an answer to the following question: should the value of the variable be considered as an integer, a real number, a character, or maybe even something else?
  – For a computer, everything is stored as a series of ones and zeros
  – We need a way to tell what the ones and zeros represent and how they should be interpreted by the program
Types in Java

- There are two broad kinds of types in Java: *primitive types* and *reference types*
  - We will see reference types later in the course
- Primitive types represent very basic kinds of values
- They are defined by the Java language and directly supported by the compiler
Primitive Types

- There are exactly 8 primitive types in Java
- Four of them represent integers (positive and negative whole numbers):
  - byte, short, int, long
- Two of them represent floating point numbers (positive and negative numbers with decimal parts):
  - float, double
- One of them represents characters:
  - char
- And one of them represents boolean values (true or false):
  - boolean
### Numeric Primitive Types

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Space</th>
<th>Literal</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>5</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>5</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>5</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>5L</td>
<td>∼ -9 \times 10^{18}</td>
<td>∼ 9 \times 10^{18}</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>5.2f</td>
<td>+/- 3.4 \times 10^{38} (with 7 significant digits)</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>5.2</td>
<td>+/- 1.7 \times 10^{308} (with 15 significant digits)</td>
<td></td>
</tr>
</tbody>
</table>
Real vs. Floating Point

• A real number is a number that can be given by an infinite decimal representation
  – For example, $\pi = 3.14159265...$
  – However, we would need infinite memory to store a number with infinite decimal representation

• The solution: floating point numbers
  – A floating point number is an approximation of a real number; it can only have a finite number of decimal places
  – Floating point numbers need only finite space (they fit in a memory cell or a finite set of memory cells)
Characters

• A `char` variable stores a single character from the Unicode character set
  ```java
  char gender;
  gender = 'M';
  ```

• A `character set` is an ordered list of characters, and each character corresponds to a unique number

• The Unicode character set uses 16 bits (2 bytes) per character, allowing for 65536 unique characters

• It is an international character set, containing symbols and characters from many world languages

• Character values are delimited by single quotation marks:
  ```
  'a'    'X'    '7'    '$'    ','    '\n'
  ```
More on Characters

- The *ASCII character set* is older and smaller than Unicode, but is still quite popular.
- The ASCII characters are a subset of the Unicode character set, including:

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper-case letters</td>
<td>A, B, C, ...</td>
</tr>
<tr>
<td>lower-case letters</td>
<td>a, b, c, ...</td>
</tr>
<tr>
<td>punctuation</td>
<td>period (.), semicolon (;), ...</td>
</tr>
<tr>
<td>digits</td>
<td>0, 1, 2, ...</td>
</tr>
<tr>
<td>special symbols</td>
<td>&amp;,</td>
</tr>
<tr>
<td>control characters</td>
<td>carriage return, line feed, tab, ...</td>
</tr>
</tbody>
</table>
Boolean Values

• An expression that evaluates either to true or to false
• Named after George Boole, the inventor of Boolean algebra
• Similar concept in natural (human) languages:
  – "the traffic light is red"
  – This expression is either true or false
Boolean Type

• You can declare variables which are of type boolean
• A boolean variable represents a true or false condition
• A boolean can also be used to represent any two states, such as a light bulb being on or off
• The literals true and false are the only valid values for a boolean type
  – While not technically reserved words, you cannot use them for any other purpose than as a literal boolean value
• Example:
  
  ```java
  boolean done = false;
  // Some code here
  done = true;
  ```
The Assignment Statement (1)

- To store a value in a variable, we use the assignment statement
- The assignment operator in Java is the = sign
- The syntax for the assignment statement is the following:
  \[ \text{variable} = \text{expression}; \]
  - The expression on the right of the = sign is evaluated
  - An expression can consist of a single variable, or something more complex involving many operators and operands
  - The result of evaluating the expression is stored in the variable on the left of the = sign; the previous value stored in that variable (if any) is overwritten
  - The values stored in the variables on the right side of the assignment statement do not change (except when special operators are used)
The Assignment Statement (2)

– The left-hand side of the $=$ MUST be a SINGLE variable; it CANNOT be an expression with multiple operators and operands
– The $=$ operator does NOT define an equation
Reading from the Keyboard (1)

- To read values from the keyboard, we use `Scanner`
- First, we must declare a variable of type `Scanner`, and initialize it
  
  ```java
  Scanner variable = new Scanner(System.in);
  ```
  
  - `variable` can be replaced by any identifier; keyboard is a good choice, but there is nothing special about it
  - Everything else, however, must be exactly as above
  - This only needs to be done once in your program, but it needs to be done before you attempt to read any values

- To read a value of type `int` from the keyboard, we use the following expression:
  
  ```java
  variable.nextInt()
  ```
Reading from the Keyboard (2)

- `variable` must be replaced by the identifier you used when you declared the `Scanner` variable
- Thus, if you called your `Scanner` variable `foo`, you must write `foo.nextInt()` if you want to read an `int` value from the keyboard

  - The expression `variable.nextInt()` (where `variable` is a variable of type `Scanner`) can be used as the right side of an assignment statement:
    ```java
    int input;
    input = keyboard.nextInt();
    ```
- The right side of the assignment statement is evaluated; this results in a value of type `int` being read from the keyboard
Reading from the Keyboard (3)

- The value read from the keyboard is then stored in the variable on the left side of the assignment statement, overwriting any previous values stored in that variable

- To read a value of type `double` instead of type `int`, simply replace `nextInt()` by `nextDouble()`
Displaying to the Monitor

- You can use one of two built-in commands to display something to the monitor:
  - `System.out.println(stuffToDisplay);`
    - Displays `stuffToDisplay` followed by a line break
  - `System.out.print(stuffToDisplay);`
    - Only `stuffToDisplay` is displayed
More on `println()` and `print()`

- `println()` and `print()` each take one input (also called parameter or argument)
  - a character string:
    - `println("Hello world!");`
  - the value of a variable:
    - `println(output);`
  - the combination of both:
    - `println("The sum is " + output);`
  - a combination can have more than two parts:
    - `println("The sum is " + sum + " and the " + " difference is " + diff);`
Examples of Displaying Text

• System.out.println("Hello world!");
  – The character string Hello world! is displayed

• int value = 5;
  System.out.println(value);
  – The value 5 is displayed

• double price = 44.99;
  System.out.println("This book costs " + price + " dollars");
  – The character string This book costs 44.99 dollars is displayed
public class Countdown {
    public static void main(String[] args) {
        System.out.print("Three... ");
        System.out.print("Two... ");
        System.out.print("One... ");
        System.out.println("Zero... ");

        System.out.println("Liftoff!");

        System.out.println("Houston, we have a problem!");
    }
}
Countdown Result

- The following is displayed when the Countdown program is executed
  - Three... Two... One... Zero... Liftoff!
  - Houston, we have a problem!

Cursor ends up here
Aside: Calling Methods (1)

• When you read a value from the keyboard, what you are really doing is calling a method
  – The method is called `nextInt()` or `nextDouble()`, depending on what you are reading
  – The method is invoked on an object which belongs to class `Scanner`
  – The name of this object is the name you gave to your `Scanner` variable

• Likewise, when you display something to the monitor, you are also calling a method
  – The method is called `println()` or `print()`
  – The method is invoked on an object which belongs to class `PrintStream`
Aside: Calling Methods (2)

- The name of this object is `System.out`

- We will revisit this aspect of reading and displaying when we cover classes and objects
Reading and Displaying: Exercise

• Write a Java program which consists of one class called ReverseDisplay. This class must define a method called main() which does the following:
  – Displays a message asking the user to enter an integer
  – Reads this integer from the keyboard
  – Displays a message asking the user to enter another integer
  – Reads this second integer from the keyboard
  – Displays the second integer entered by the user along with a message describing the meaning of this value
  – Displays the first integer entered by the user along with a message describing the meaning of this value
Program Template

• To make it easier to write programs, you can use the following template as a starting point:

```java
import java.util.Scanner;

public class Template {
    // Change "Template" to the desired name
    // for your class
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        // Write your program statements here
    }
}
```
Part 3: Expressions
Arithmetic Expressions

• An expression is a combination of operators and operands
  – Operators are optional but operands are mandatory
  – An operand can be a literal value (for example, 5 or 3.14), a variable, or the value returned by a method call (like `nextInt()`)  

• Arithmetic expressions compute numeric results and make use of the arithmetic operators:
  
  Addition: \( x + y \)
  Subtraction: \( x - y \)
  Multiplication: \( x \times y \)
  Division: \( x / y \)
  Remainder: \( x \mod y \)
  Negation: \( -x \)
Division With Integers

• If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)
• The remainder operator (%) returns the remainder after dividing the second operand into the first
• Example:

```java
int numHours = 52;
int fullDays = numHours / 24;
// fullDays contains 2
int remainingHours = numHours % 24;
// remainingHours contains 4
```

• Division by 0 with integers
  – Produces run-time error
  – Program has to avoid it, or it will crash
DivisionInt.java

```java
import java.util.Scanner;

public class DivisionInt {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int numerator, denominator, quotient, remainder;

        // Read the values
        System.out.print("Enter the numerator: ");
        numerator = keyboard.nextInt();
        System.out.print("Enter the denominator: ");
        denominator = keyboard.nextInt();

        quotient = numerator / denominator;
        remainder = numerator % denominator;
        System.out.println("The result is: "+quotient);
        System.out.println("The remainder is: "+remainder);
    }
}
```

What does this display?
Operator Precedence

• Operators can be combined into complex expressions:
  
  \[ \text{result} = \text{total} + \frac{\text{count}}{\text{max}} - \text{offset}; \]

• Operators have a well-defined precedence which determines the order in which they are evaluated:

• Multiplication (\(\times\)), division (\(\div\)), and remainder (\(\%\)) are evaluated prior to addition (\(+\)) and subtraction (\(-\))

• Arithmetic operators with the same precedence are evaluated from left to right

• Parentheses can always be used to force the evaluation order
Operator Precedence Examples

- What is the order of evaluation in the following expressions?

\[
\begin{align*}
\text{a + b + c + d + e} & \quad \text{a / (b + c) - d % e} \\
1 & \quad 2 \\
2 & \quad 1 \\
3 & \quad 4 \\
4 & \quad 3
\end{align*}
\]

\[
\begin{align*}
\text{a / (b + c) - d % e} & \quad \text{a / (b * (c + (d - e)))} \\
2 & \quad 1 \\
4 & \quad 3 \\
3 & \quad 2 \\
1 & \quad 4
\end{align*}
\]
Assignment Operator Precedence

- The assignment operator has a lower precedence than the arithmetic operators

```
answer = sum / 4 + MAX * lowest;
```

First, the expression on the right side of the `=` operator is evaluated.

Then the result is stored in the variable on the left-hand side.
Assignment Operator Sides

- The left-hand and right-hand sides of an assignment statement can contain the same variable:

  \[ \text{count} = \text{count} + 1; \]

  First, 1 is added to the original value of \text{count}; the result is stored in a temporary memory location.

  Then, the overall result is stored into \text{count}, overwriting the original value.

- The fact that the assignment operator has lower precedence than arithmetic operators allows us to do this.
Arithmetic Expressions: Exercise 1

• Write a Java program which consists of one class called AddTwoIntegers. This class must define a method called main() which does the following:
  – Displays a message asking the user to enter an integer
  – Reads an integer from the keyboard
  – Displays a message asking the user to enter another integer
  – Reads another integer from the keyboard
  – Adds the two integers read from the keyboard together
  – Displays the result with an appropriate message

• How would you modify this program to add two real numbers instead?
Arithmetic Expressions: Exercise 2

• Write a Java program which consists of one class called `TemperatureConverter`. This class must define a method called `main()` which does the following:
  – Displays a message asking the user to enter a temperature in Celcius degrees
  – Converts the temperature in Celcius degrees to Fahrenheit degrees; the formula for this calculation is \( f = \left( \frac{9c}{5} \right) + 32 \), where \( f \) is the temperature in Fahrenheit, and \( c \) is the temperature in Celcius
  – Displays the equivalent temperature in Fahrenheit with an appropriate message
Increment / Decrement Operators

- The increment and decrement operators are arithmetic and operate on one operand
  - This operand must be a single variable
- The *increment operator* (++) adds one to its operand
- The *decrement operator* (--) subtracts one from its operand
- The increment and decrement operators can be applied in *prefix form* (before the variable) or *postfix form* (after the variable)
Increment / Decrement Semantics

- When used alone in a statement, the prefix and postfix forms are basically equivalent
- That is, when used alone in a statement,
  
  ```
  ++count;
  ```

  is equivalent to
  
  ```
  count++;
  ```

  which is equivalent to
  
  ```
  count = count + 1;
  ```

- However, *they are not equivalent when they are used in expressions!*
  - Using these operators in expressions requires you to really know what you are doing
Increment / Decrement Advice

• Using increment and decrement operators is fine if this operator is the only operation in a statement
  
  i++; // OK
  ++j; // OK too

• Do not use increment and decrement operators in more complex statements
  
  total = count++ - --count; // Avoid!
  index = ++i * --j / k++; // Avoid too!
Assignment Operators (1)

- Often we perform an operation using a variable, then store the result back into that variable
- Java provides additional assignment operators to simplify that process
  - They combine the assignment operator with an arithmetic operator
- Example 1: The statement
  
  ```java
  total += 5;
  ```
  is equivalent to
  
  ```java
  total = total + 5;
  ```
- Example 2: The statement
  
  ```java
  result *= count1 + count2;
  ```
  is equivalent to
  
  ```java
  result = result * (count1 + count2);
  ```
Assignment Operators (2)

- In general, assignment operators have the form
  \[ \text{variable op= expression;} \]
  which is equivalent to
  \[ \text{variable} = \text{variable op (expression);} \]

- The entire expression on the right side of the assignment operator is evaluated
  - It can be a complex expression involving many levels of parentheses

- Then, the result of evaluating that expression is used on the right hand side of the operator in the assignment operator
  - The left operand is the variable on the left of the assignment operator

- The final result is stored back in the variable on the left-hand side of the assignment operator
Assignment Operators (3)

- There are many assignment operators, including the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += y;</td>
<td>x = x + y;</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y;</td>
<td>x = x - y;</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y;</td>
<td>x = x * y;</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y;</td>
<td>x = x / y;</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y;</td>
<td>x = x % y;</td>
</tr>
</tbody>
</table>
Boolean Expressions

• Instead of evaluating to a numeric value, boolean expressions evaluate to either true or false
  myNumber > 0 // can be either true or false

• You can assign the result of a boolean expression to a variable of type boolean:
  boolean positive;
  positive = (myNumber > 0);

• Boolean expressions are often used to control the flow of execution of a program
  – We will see control flow in detail later in the course; we will cover boolean expressions in more detail then
Constants

- A constant is an identifier that is similar to a variable except that it holds one value for its entire existence.

- In Java, we use the `final` modifier to declare a constant:
  ```java
  final double PI = 3.14;
  ```

- The compiler will issue an error if you try to assign a value to a constant more than once in the program:
  ```java
  final double PI = 3.14;
  // Some more statements...
  PI = 2.718;
  // Error: cannot assign a value to a // final variable more than once
  ```

- Java style conventions for constants: identifiers for constants should contain only upper-case letters, and each word in the identifier should be separated by an underscore (_).
Advantages of Constants

• Constants give names to otherwise unclear literal values
  – The meaning of 0.05 may not be clear to the person reading your code, but the meaning of GST or TAX is

• Constants facilitate changes to the code
  – More precision required? Change PI from 3.14 to 3.14159265 in only one place
  – No need to search the whole program for occurrences of the value

• Constants prevent inadvertent errors
  – The programmer cannot type the wrong value by mistake
  – If you type Pi instead of PI, the compiler will report an error; if you type 31.4 instead of 3.14, the compiler will not report an error but your calculations will probably be all wrong
Constants: Exercise

• Write a Java program which consists of one class called Circle. This class should define a method called main() which does the following:
  – Displays a message asking the user to enter the radius of a circle, and reads the value of the radius from the keyboard
  – Computes the circumference of the circle; the formula for this calculation is \( c = 2\pi r \), where \( r \) is the radius of the circle and \( c \) is its circumference
  – Computes the area of the circle; the formula for this calculation is \( a = \pi r^2 \), where \( r \) is the radius of the circle and \( a \) is its area
  – Displays both values with an appropriate message

• You must declare \( \pi \) as a constant called \( \text{PI} \), with 4 digits after the decimal, and use this constant in your calculations
Part 4: Data Conversion
Data Conversion

• Sometimes it is convenient to convert data from one type to another

• For example, we may want to treat an integer as a floating point value during a computation

• Conversions must be handled carefully to avoid losing information

• There are two types of conversions:
  – Widening conversions
  – Narrowing conversions

• But first, a word on precision
Precision

• The *precision* of a type is the range of all possible values you can express with a variable of that type
  – Variables of type `int` have higher precision than variables of type `short`; you can express more values with a variable of type `int` than with a variable of type `short`
  – Variables of type `double` have higher precision than variables of type `int`; again, you can express more values with a variable of type `double` than with a variable of type `int`

• There is a correlation between the number of bytes used to store a value of a given type, and the precision of that type
  – `byte` uses only 8 bits and has the lowest precision; `double` uses 64 bits and has the highest precision
Conversion Types

• *Widening conversions* occur when a value whose type has lower precision is converted to a type that has higher precision (such as a `short` to an `int` or an `int` to a `double`)
  – They are *usually* safe, as there is *usually* no information lost

• *Narrowing conversions* occur when a value whose type has higher precision is converted to a type that has lower precision (such as an `int` to a `short` or a `double` to `int`)
  – Information can be lost when conversions of this type occur
Assignment Conversion (1)

• In Java, data conversions can occur in three ways:
  – Assignment conversion
  – Arithmetic promotion
  – Casting

• Assignment conversion occurs when a value of one type is assigned to a variable of another type
  – Only widening conversions can occur via assignment (such as assigning a value of type `int` to a variable of type `double`)
  – If we attempt a narrowing conversion via assignment (such as assigning a value of type `double` to a variable of type `int`), the compiler will issue an error
Assignment Conversion (2)

• Assignment conversions occur automatically
  – For example, if we attempt to assign the value of an expression of type `int` to a variable of type `double`, the value of the expression will be automatically converted to have type `double`

• The second assignment statement below is perfectly legal:
  ```java
  int i = 7;
  double d = 3 + i;
  ```
  The expression on the right-hand side has type `int`, which has lower precision than type `double`, the type of the variable on the left-hand side

• The value of `3 + i` gets converted to have type `double` and gets assigned to `d`; note that the type of `i` does not change
Assignment Conversion (3)

• The assignment second assignment statement below is illegal:
  
  ```java
  double d = 7.0;
  int i = 3.0 + d;
  ```

  The expression on the right-hand side has type `double`, which has higher precision than type `int`, the type of the variable on the left-hand side

• The compiler will therefore report an error
Arithmetic Promotion

- *Arithmetic promotion* happens automatically when operators in expressions convert their operands

```java
double kmPerLitre;
int km = 1000;
float litres = 85.0f;
kmPerLitre = km / litres;
```

1. Arithmetic promotion to float
2. Division
3. Assignment conversion to double
4. Assign result to kmPerLitre
Casting (1)

- *Casting* (also called *typecasting*) is the most powerful, general, and dangerous, technique for conversion
- It trusts that you know what you are doing
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
  - It is the only way to perform a narrowing conversion
- To cast, the type you want to convert a value to is put in parentheses in front of the value being converted. The general syntax is thus:
  \[(\text{desiredType}) \text{ expression}\]
- The cast does not change the type of the value in a variable for the rest of the program, only for the operation in which the value is cast
Casting (2)

• When casting from floating point to integer, the fractional part of the floating point is discarded

```java
double money = 25.80;
int dollars;

dollars = (int)money;
// dollars contains the value 25
// money still contains the value 25.80
```
Casting (3)

- Casting has higher precedence than the arithmetic operators

```java
double a;
int b, c, d;

a = 3.5;
b = 3;

c = (int)a * b;  // Only the value of a is cast; the value of b is not cast
Thus, c now contains the value 9

\[ d = (int)(a \times b); \]  // The product of a and b is cast
Thus, d now contains the value 10
```
Conversion Traps (1)

• Consider the following variable declarations:
  ```java
  int total = 10;
  int count = 4;
  double result;
  ```

• What will the value of `result` be after the following statement is executed?
  ```java
  result = total / count;
  ```

• The value of `result` is now `2.0` (!?!)?

• The two operands of the division operator have type `int`; thus, integer division is performed, truncating the fractional part
  – The result of `total / count` is therefore `2` and has type `int`
Conversion Traps (2)

- It's only when the quotient is assigned to `result` that it is converted to type `double` via assignment conversion.

- Casting one of the integer operands to `double` (or `float`) will result in floating point division being performed instead:
  ```
  result = (float)total / count;
  ```

- The value of `result` is now 2.5
  - First, `total` is cast to type `float`.
  - Arithmetic promotion occurs to promote `count` to have type `float`; floating point division is performed and the result of the division therefore has type `float`.
  - The quotient is converted to type `double` before being assigned to `result`.
## Conversion Examples

<table>
<thead>
<tr>
<th>Code Fragment</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>double x = 5.9; int y = (int) x;</td>
<td>y contains value 5</td>
</tr>
<tr>
<td>int a = 5; float b = 7.3; double c = 10.03; c = b + a;</td>
<td>c contains value 12.3</td>
</tr>
<tr>
<td>int a = 2, b = 5; double c = 22; c = a / b;</td>
<td>c contains value 0.0</td>
</tr>
</tbody>
</table>
Part 5: Problem Solving
Problem Solving

• The purpose of writing a program is to solve a problem (or rather, have the computer solve a problem for us)

• The general steps in problem solving are:
  – Understand the problem
  – Dissect the problem into manageable pieces
  – Design a solution for each of the pieces
  – Consider alternatives to the solutions and refine them
  – Implement the solutions
  – Test the solutions and fix any problems that exist
  – Combine the solutions for each of the piece to obtain the solution to the original problem
Part 6: Exercises
Exercises (1)

1. Write a program which consists of a single class called **BMICalculator**. This class must define a method called **main()** which does the following:

   • Asks the user to enter his/her weight (in kilograms) and his/her height (in meters)
   • Calculates the user's Body Mass Index (BMI); a person's BMI can be computed by dividing his /her weight in kilograms by the square of his / her height in meters.
   • Displays the user's BMI it to the screen
Exercises (2)

2. Write a program which consists of a single class called EnergyCalculator. This class defines a method called main() which does the following:

- Asks the user to enter a quantity of water (in kilograms), an initial temperature (in Celsius degrees), and a final temperature (also in Celsius degrees)
- Calculates how much energy (in joules) is needed to raise that quantity of water from the initial temperature to the final temperature; the formula for this calculation is \( q = 4184 \cdot m \cdot (f - i) \), where \( q \) is the energy, \( m \) is the quantity of water, \( f \) is the final temperature, and \( i \) is the initial temperature. You may assume that there are no state transitions (from ice to water or water to steam, for example) in the temperature interval
- Displays the required energy quantity to the screen
3. Write a program which consists of a single class called GoalieStatsCalculator. This class defines a method called main() which does the following:

- Asks the user to enter the number of minutes an ice hockey goalie has played, the number of goals he/she allowed, and the total number of shots he/she faced
- Calculates the goalie's goals against average (GAA) and save percentage; the GAA is the number of goals allowed per 60 minutes of play, while the save percentage is the number of saves (the total number of shots minus the number of goals allowed) divided by the number of shots faced
- Displays the goalie's GAA and save percentage to the screen