Software Engineering1
(Java)

CSY1019
(Week 2)

Sample Program

```
// Display a greeting in the console window

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

//Recap: How to use Eclipse and set up workspace [Guides on NILE]

Parts of a Java Program

- Comments
  - The line is ignored by the compiler.
  - The comment in the example is a single-line comment.
- Class Header
  - The class header tells the compiler things about the class such as what other classes can use it (public) and that it is a Java class (class), and the name of that class (Simple).
- Curly Braces
  - When associated with the class header, they define the scope of the class.
  - When associated with a method, they define the scope of the method.

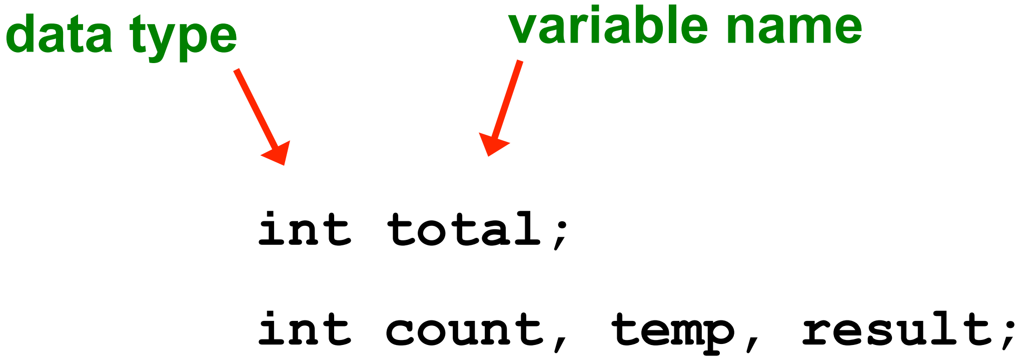
Short Review

- Java is a case-sensitive language.
- All Java programs must be stored in a file with a .java file extension.
- Comments are ignored by the compiler.
- A .java file may contain many classes but may only have one public class.
- If a .java file has a public class, the class must have the same name as the file.

Variables

- A *variable* is a name for a location in memory

- A variable must be *declared* by specifying its name and the type of information that it will hold



data type

variable name

```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used
- See example: Variable.java

Variables and Literals

This line is called a *variable declaration*.

```
int value;
```

The following line is known as an assignment statement.

```
value = 5;
```

0x000	
0x001	5
0x002	
0x003	

The value 5 is stored in memory.

This is a string *literal*. It will be printed as is.

```
System.out.print("The value is ");
System.out.println(value);
```

The integer 5 will be printed out here.
Notice no quote marks?

The + Operator

- The + operator can be used in two ways.
  - as a concatenation operator
  - as an addition operator
- If either side of the + operator is a string, the result will be a string.

```
System.out.println("Hello " + "World");
System.out.println("The value is: " + 5);
System.out.println("The value is: " + value);
System.out.println("The value is: " + 5 + 3 );
System.out.println("The value is: " + (5 + 3));
```

Identifiers

- Identifiers are programmer-defined names for:
  - classes
  - variables
  - methods
- Identifiers may not be any of the Java reserved keywords.

Identifiers

- Identifiers must follow certain rules:
  - An identifier may only contain:
    - letters a–z or A–Z,
    - the digits 0–9,
    - underscores (_), or
    - the dollar sign ($)
  - The first character may not be a digit.
  - Identifiers are case sensitive.
    - `itemsOrdered` is not the same as `itemsordered`.
  - Identifiers cannot include spaces.

Variable Names

- Variable names should be descriptive.
- Descriptive names allow the code to be more readable; therefore, the code is more maintainable.
- Which of the following is more descriptive?

```
double tr = 0.0725;
double salesTaxRate = 0.0725;
```

- Java programs should be *self-documenting*.

Java Naming Conventions

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

  Ex: `int caTaxRate`

- Class names should be all title case.

  Ex: `public class BigLittle`

- More Java naming conventions can be found at:

  http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

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

Primitive Data Types

- There are eight primitive data types in Java
- Four of them represent integers
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers
  - `float (e.g. float num = 23.5F;)`
  - `double (e.g. double num = 14520.904;)`
- One of them represents characters
  - `char    (e.g. char letter = ‘a’;)`
- And one of them represents boolean values
  - `Boolean (e.g. boolean flag = true;)`

Numeric Primitive Data

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

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	< -9 x 10^{18}	> 9 x 10^{18}
`float`	32 bits	+/- 3.4 x 10^{38} with 7 significant digits	
`double`	64 bits	+/- 1.7 x 10^{308} with 15 significant digits	
`char`	16 bits		
`boolean`	1 bit		

Variable Assignment and Initialization

- In order to store a value in a variable, an *assignment statement* must be used.
- The *assignment operator* is the equal (=) sign.
- The operand on the left side of the assignment operator must be a variable name.
- The operand on the right side must be either a literal or expression that evaluates to a type that is compatible with the type of the variable.

Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                                                                    days + " Days.");
  }
}
```

The variables must be declared before they can be used.

Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                        days + " Days.");
  }
}
```

Once declared, they can then receive a value (initialization); however the value must be compatible with the variable’s declared type.

Variable Assignment and Initialization

```java
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                        days + " Days.");
  }
}
```

After receiving a value, the variables can then be used in output statements or in other calculations.

Variable Assignment and Initialization

```
// This program shows variable initialization.

public class Initialize
{
  public static void main(String[] args)
  {
    int month = 2, days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

Local variables can be declared and initialized on the same line.

Variable Assignment and Initialization

- Variables can only hold one value at a time.
- Local variables do not receive a default value.
- Local variables must have a valid type in order to be used.

```
public static void main(String [] args)
 {
   int month, days; //No value given…
  System.out.println("Month " + month + " has " +
                            days + " Days.");
 }
```

Trying to use uninitialized variables will generate a Syntax Error when the code is compiled.

Integer Data Types

- `byte`, `short`, `int`, and `long` are all integer data types.
- They can hold whole numbers such as 5, 10, 23, 89, etc.
- Integer data types cannot hold numbers that have a decimal point in them.
- Integers embedded into Java source code are called *integer literals*.
- See Example: IntegerVariables.java

Floating Point Data Types

- Data types that allow fractional values are called *floating-point* numbers.
  - 1.7 and -45.316 are floating-point numbers.
- In Java there are two data types that can represent floating-point numbers.
  - `float` - also called *single precision* (7 decimal points).
  - `double` - also called *double precision* (15 decimal points).

Floating Point Literals

- When floating point numbers are embedded into Java source code they are called *floating point literals*.
- The default type for floating point literals is `double`.
  - 29.75, 1.76, and 31.51 are `double` data types.
- See example: Sale.java

Floating Point Literals

- A `double` value is not compatible with a `float` variable because of its size and precision.
  - `float number;`
  - `number = 23.5; // Error!`
- A `double` can be forced into a `float` by appending the letter F or f to the literal.
  - `float number;`
  - `number = 23.5F; // This will work.`

The `boolean` Data Type

- The Java `boolean` data type can have two possible values.
  - `true`
  - `false`
- The value of a `boolean` variable may only be copied into a `boolean` variable.

See example: TrueFalse.java

The `char` Data Type

- The Java `char` data type provides access to single characters.
- `char` literals are enclosed in single quote marks.
  - ‘a’, ‘Z’, ‘\n’, ‘1’
- Don’t confuse `char` literals with string literals.
  - `char` literals are enclosed in single quotes.
  - String literals are enclosed in double quotes.

See example: Letters.java

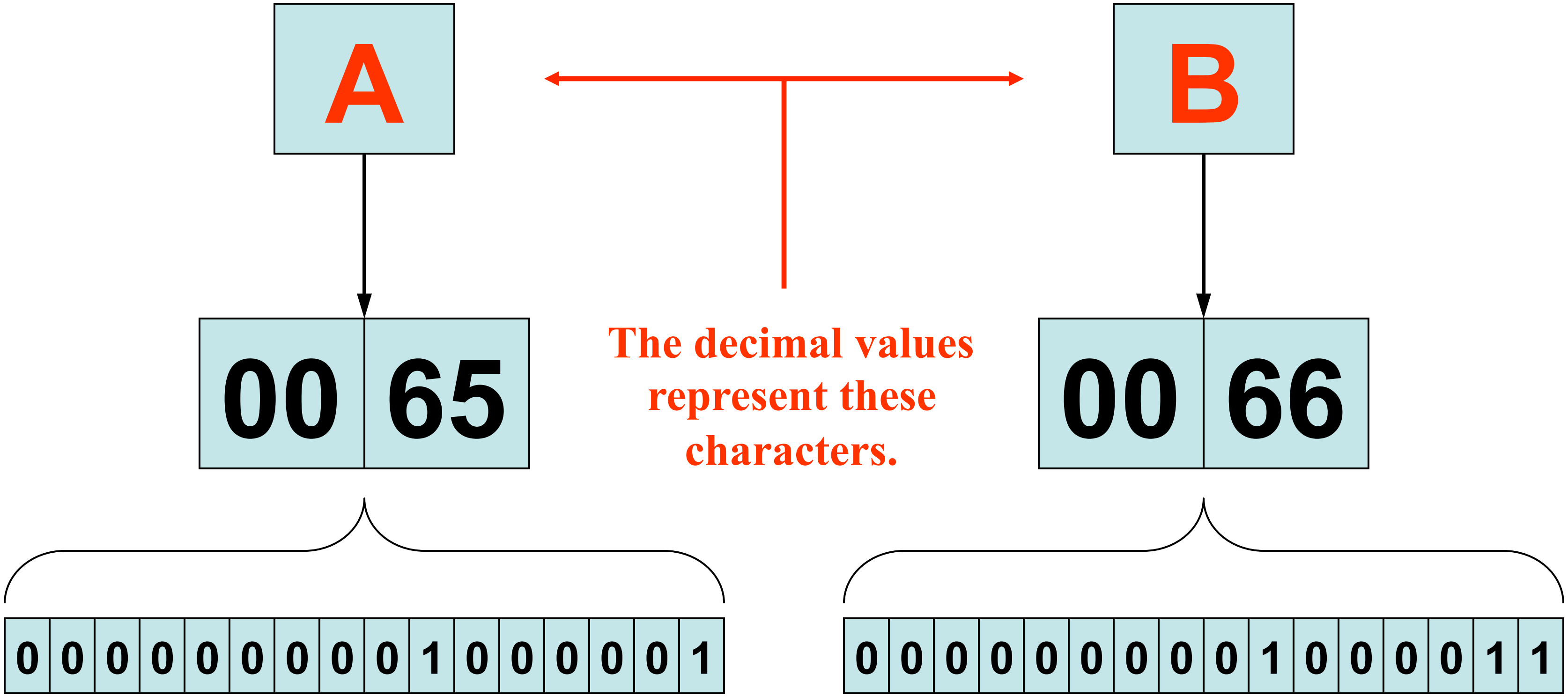
Unicode

- Internally, characters are stored as numbers.
- Character data in Java is stored as Unicode characters.
- The Unicode character set can consist of 65536 (2^{16}) individual characters.
- This means that each character takes up 2 bytes in memory.
- The first 256 characters in the Unicode character set are compatible with the ASCII* character set.

See example: Letters2.java

*American Standard Code for Information Interchange

Unicode



A

B

00	65

00	66

The decimal values represent these characters.

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1

Arithmetic Operators

- Java has five (5) binary arithmetic operators.

Operator	Meaning	Type	Example
+	Addition	Binary	`total = cost + tax;`
-	Subtraction	Binary	`cost = total - tax;`
*	Multiplication	Binary	`tax = cost * rate;`
/	Division	Binary	`salePrice = original / 2;`
%	Modulus	Binary	`remainder = value % 5;`

Arithmetic Operators

- The operators are called binary operators because they must have two operands.
- Each operator must have a left and right operand.

  See example: Wages.java
- The arithmetic operators work as one would expect.
- It is an error to try to divide any number by zero.
- When working with two integer operands, the division operator requires special attention.

Integer Division

- Division can be tricky.

  In a Java program, what is the value of 1/2?
- You might think the answer is 0.5…
- But, that’s wrong.
- The answer is simply 0.
- Integer division will truncate any decimal remainder.

A Closer Look at the / Operator

- `/` (division) operator performs integer division if both operands are integers

```
X = 13 / 5;     // X = 2
Y = 91 / 7;     // Y = 13
```

- If either operand is floating point, the result is floating point

```
X = 13 / 5.0;   // X = 2.6
Y = 91.0 / 7;   // Y = 13.0
```

A Closer Look at the `%` Operator

- `%` (modulus) operator computes the remainder resulting from integer division

```
a = 13 % 5;    // a = 3
```

Precedence

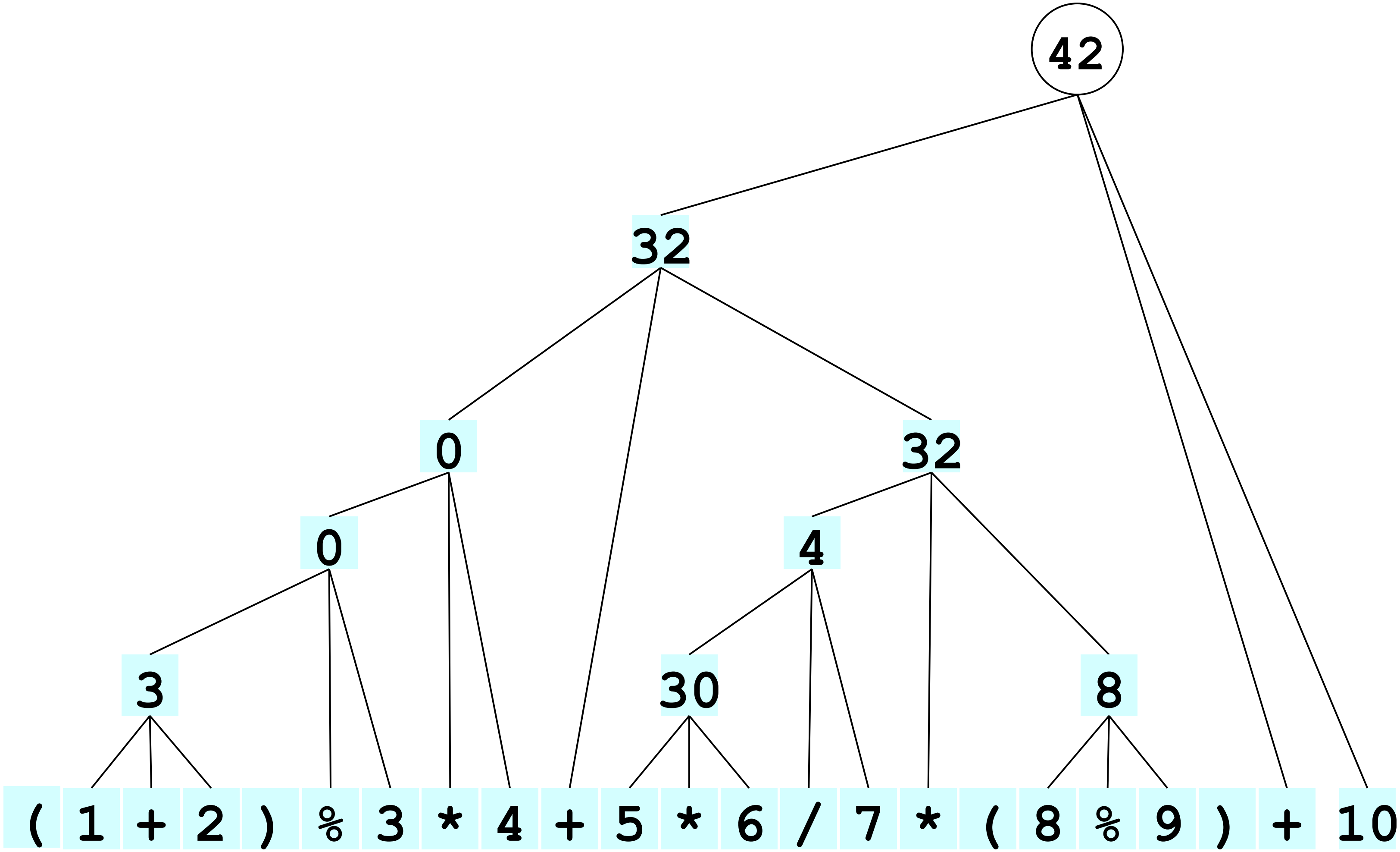
- If an expression contains more than one operator, Java uses **precedence rules** to determine the order of evaluation. The arithmetic operators have the following relative precedence:

Parentheses ( )	*highest*
unary −	↕
`*` `/` `%`	
`+` `−`	*lowest*

Operator	Associativity	Example	Result
− (unary negation)	right to left	`x = -4 + 3;`	-1
`* / %`	left to right	`x = -4 + 4 % 3 * 13 + 2;`	11
`+ -`	left to right	`x = 6 + 3 - 4 + 6 * 3;`	23

Exercise: Precedence Evaluation

What is the value of the expression at the bottom of the screen?



```
( 1 + 2 ) % 3 * 4 + 5 * 6 / 7 * ( 8 % 9 ) + 10
```

Exercises: What is the answer in Java?

- 12 + 6/ 3 = 14
- ((4 * 5) / (5-2)) – 25 = -19
- 3 * 3 + 5 % 2 = 10
- 3 + 2/5 + -2 * 4 = -5
- 2 * (1 – (3/4) / 2) * (2 – 6 % 3) = 4
- 4 + 11/2.0 – (32 % 4) + 5 - 25 = -10.5
- 8 * (32 - 2) / 12 – (5/7) + (11.0/5.0) = 22.2
- 4 + 22 % 2 - (22 – 21) + (25/2) + 4 - 3.0/2 = 17.5

The Class `String`

- We've used constants of type `String` already.

  `"Enter a whole number from 1 to 99."`

- A value of type `String` is a
  - Sequence of characters
  - Treated as a single item.

String Constants and Variables

- Declaring

```
String greeting;
 greeting = "Hello!";
```

or

```
String greeting = "Hello!";
```

or

```
String greeting = new String("Hello!");
```

- Printing

```
System.out.println(greeting);
```

Example: **StringDemo.java**

The `Scanner` Class

- The `Scanner` class provides convenient methods for reading input values of various types
- A `Scanner` object can be set up to read input from a source, including the user typing values on the keyboard
- Keyboard input is represented by the `System.in` object

The `Scanner` Class

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

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

Reading Input

- The following line creates a Scanner object that reads from the keyboard

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

- The `new` operator creates the `Scanner` object

- Once created, the `Scanner` object can be used to invoke various input methods, such as

```
   String answer = scan.nextLine();
```

Reading Input

```
String line = scan.nextLine(); // for a line of text
String word = scan.next(); // for a word
char character = scan.next().charAt(0)// for character
int integer = scan.nextInt();//for integer
double number = scan.nextDouble();//for double
```

....

and so on …

The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used. (use import java.util.Scanner)

Example Programs (using Scanner Class)

See:

ScannerDemo.java

Payroll.java

GasMileage.java

Scope

- *Scope* refers to the part of a program that has access to a variable's contents.
- Variables declared inside a method (like the main method) are called *local variables*.
- Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.

See example: Scope.java (This program contains an intentional error.)

Programming Style

- Although Java has a strict syntax, whitespace characters are ignored by the compiler.
- The Java whitespace characters are:
  - space
  - tab
  - newline
  - carriage return
  - form feed

See example: Compact.java

Indentation

- Programs should use proper indentation.
- Each block of code should be indented a few spaces from its surrounding block.
- Two to four spaces are sufficient.
- Tab characters should be avoided.
  - Tabs can vary in size between applications and devices.
  - Most programming text editors allow the user to replace the tab with spaces.

See example: Readable.java