Programming Techniques Part 1- JAVA

BCNS1102C

Lecture 3: Operators and Expression

Lecturer: Mr. Ajit. Gopee

E-mail: ajit.gopee@utm.ac.mu

Outline

- Structure of a java program
- Output to the console
- Variables
- keywords
- Data Types
- Operators
- Expressions
- Oder of operators
- Operators for Strings

OBJECTIVES

In this chapter you'll learn:

- To write simple Java applications.
- To use input and output statements.
- Java's primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

First Program: Hello World

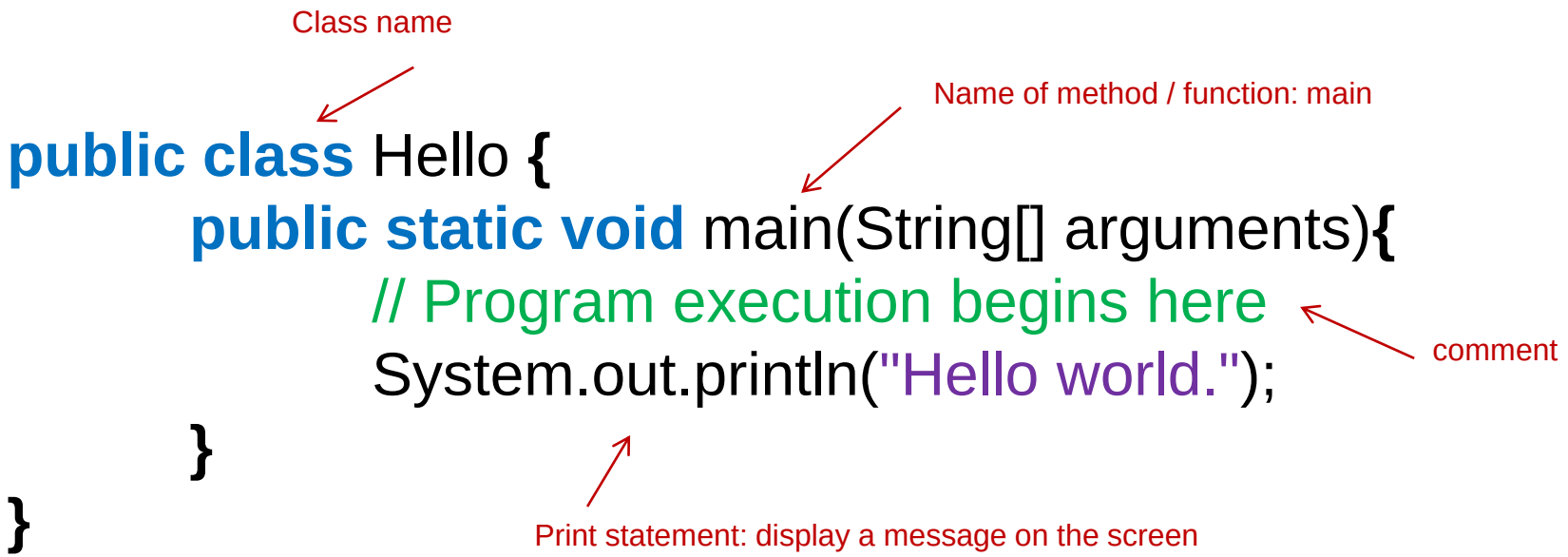
```
public class Hello {
    public static void main(String[]
    arguments){
        // Program execution begins here
        System.out.println("Hello world.");
    }
}
```

Program Structure

```
public class Classname {
    public static void main(String[] arguments) {
        Statements
    }
}
```

- Java programs are made up of class definitions, and the classname is name chosen by the programmer.
  - Hello was the name chosen in the first programme.

Structure of the Hello Program



- System.out.println is a method provided by one of Java's libraries.
  - A library is a collection of class and method definitions.
- Notice the use of curly braces **{ }** and semicolon **;**
- Comments can also be in the following form **/* comments go hear */**

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10    } // end method main
11 } // end class Welcome1
```

```
Welcome to Java Programming!
```

Fig. 2.1 | Text-printing program.

Our First Program in Java: Printing a Line of Text (Cont.)

- Comments

```
// Fig. 2.1: Welcome1.java
```

  - `//` indicates that the line is a comment.
  - Used to document programs and improve their readability.
  - Compiler ignores comments.
  - A comment that begins with `//` is an end-of-line comment—it terminates at the end of the line on which it appears.
- Traditional comment, can be spread over several lines as in

```
/* This is a traditional comment. It
   can be split over multiple lines */
```

  - This type of comment begins with `/*` and ends with `*/`.
  - All text between the delimiters is ignored by the compiler.

Our First Program in Java: Printing a Line of Text (Cont.)

- Javadoc comments
  - Delimited by `/**` and `*/`.
  - All text between the Javadoc comment delimiters is ignored by the compiler.
  - Enable you to embed program documentation directly in your programs.
  - . Forgetting one of the delimeters is a **syntax error**
  - **3 types of errors in programming: Syntax, Run-time, Logical errors** .
  - Good practice to include comments
  - All programs should start with comments.

Class names

- By convention, begin with a capital letter and capitalize the first letter of each word they include (e.g. SampleClassName)
- A class name is an identifier- a series of characters consisting of letters, digits, underscores(_), and dollar sign ($) that does not begin with a digit and does not contain spaces.
- Java is case sensitive – so a1 and A1 are different (but both valid) identifiers.

- A *public* class must be placed in a file that has the same name as the class (in terms of spelling and capitalization) plus the .java extension otherwise it is a compilation error.

- Indentation

Semicolons, Blocks, White Spaces, Comments

- A statement is one or more lines of code terminated by a semicolon.
- A block is a group of statements bound by opening and closing braces.
- Blank lines and space characters (collectively named as White Space) are used to enhance the clarity and visual appearance of the source code.
- White space is ignored by the compiler
- **Comments:**
  - // - comment on one line
  - /* …*/ - multiline comments
  - /** … */ -documentation comment

- Public static void main(String[] args) - is the starting point of every Java Program
- (main) Method
- void
- System.out.println(“Hello World”);
- Hello World here is the string (also known as character string or literal string)
- System.out - object- it allows strings to be displayed in command window.

Output to the Console

- System.out.println(“some text goes here”);
  - It outputs to the console

- Strings appears between the quotation marks
  - Sequence of characters
  - can contain any combination of letters, numbers, punctuation marks, and other special characters.

- println is short for print line
  - It adds a new line after each printed line.

println vs. print

```
public class Welcome2 {
    public static void main(String[] args){
        System.out.println(“Welcome to”);    //Print once
        System.out.println(“Java Programming”);// Again!
    }
}
```

```
public class Welcome3 {
    public static void main(String[] args){
        System.out.print(“Welcome to");       //no new line
        System.out.println(“Java Programming”);// with new
    line
    }
}
```

- Newline characters
- Backslash (\) – escape chatacter
- Backslash can be combined with the next character to form an escape sequence
- The escape sequence \n represents the new line character
- Other escape characters

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do *not* advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`.

Fig. 2.5 | Some common escape sequences.

\n escape sequence

```
public class Welcome2 {
    public static void main(String[] args){
        System.out.println("Welcome\n to\n Java\n
    Programming");
    }
}
```

Displaying text with printf

- Syste.out.printf method
- ‘f” means “formatted”- so it displays formatted data
- Multiple method arguments are placed in a comma-separated list
- printf method to format string
- May consist of fixed text or format specifiers
- Fixed text is output as it would be by print or println
- Each format specifier is a placeholder for a value and specifies the type of data to output
- Format specifiers begin with the percent sign (%) and are followed by a character that represents the data type
- Format specifier %s is a placeholder for a string.

```
1  // Fig. 2.6: Welcome4.java
2  // Displaying multiple lines with method System.out.printf.
3
4  public class Welcome4
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.printf( "%s\n%s\n",
10          "Welcome to", "Java Programming!" );
11    } // end method main
12 } // end class Welcome4
```

Each %s is a placeholder for a String that comes later in the argument list

Statements can be split over multiple lines.

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

Variables

- A variable is a named location that stores a value.
  - Values are things that can be printed, stored and operated on.
  - Strings values (Hello World, How are you?) printed are values.
- A value which is stored in a variable has a **type**.
- This is the basic format for creating a variable

  Type name;
- Examples:

  String word1;

  int num1;

Identifiers

- It is a name given to a **variable**, class, or method.

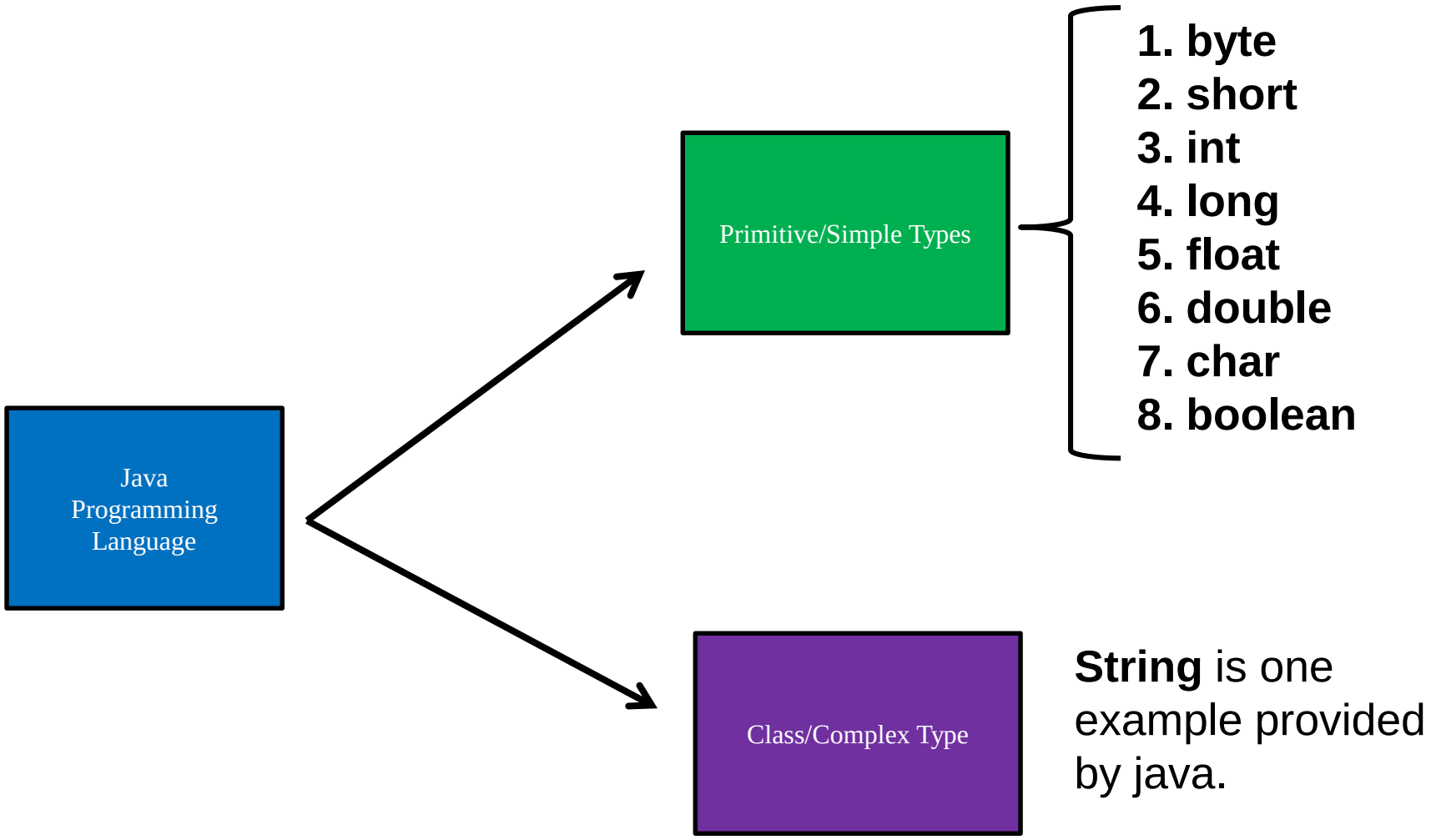
- Rules:
  - Identifier can start with a ***Unicode letter, underscore, or dollar sign***.
  - Identifiers are ***case-sensitive*** and have no maximum length.
  - An identifier cannot be a ***keyword***, but it can contain a keyword as part of its name.
  - Identifiers can use non-ASCII characters.

Keywords

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

Java Programming Language Keywords

Data Types



String is one example provided by java.

Primitive Types

- Eight Primitive Data Types (in four categories):
  - Logical Category:
    - boolean *(values: true or false)*
  - Textual Category:
    - char *(represent single characters, e.g 'A')*
  - Integral Category:
    - byte, short, int and long
  - Floating Point:
    - double and float

Assignment

- Once we declare a variable we can give it a value using the assignment operator.

```
String word1;                 // declare a variable word1 of type
    String
word1 = "Hello.";             // give word1 the value "Hello."


int hour;                     // declare a variable time of type int
    (integer)
hour = 11;                    // assign value 11 to hour


int minute = 38;              // assignment combined with declaration
```

- A variable has the same type as the value assigned to it.**

Exercise

- Which of the following assignments are legal?

String word1;
word1 = “123”;

A

String word1;
word1 = 123;

B

Answer: A

Variable word1 holds a string which is made up of character 1, 2 & 3

whereas in the other case, 123 is considered as a number(integer).

Using Variables

```
class Hello4 {
   public static void main(String[] args){
                String name;
        name = “John”;
        System.out.println(name);
        name = “Jane”;
        System.out.println(name);
   }
}
```

Constant in Java

- What is constant?
- **Constant** is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final.
- How to declare constant in Java?
- In Java, to declare any variable as constant, we use static and final modifiers. It is also known as **non-access** modifiers. According to the Java naming convention the identifier name must be in **capital letters**.

Constant in Java

- Static and Final Modifiers
- The purpose to use the static modifier is to manage the memory.
- It also allows the variable to be available without loading any instance of the class in which it is defined.
- The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.

Constant in Java - Example

- **static final** datatype identifier_name=value;
- For example, **price** is a variable that we want to make constant.
- **static final double** PRICE=432.78;
- Where static and final are the non-access modifiers. The double is the data type and PRICE is the identifier name in which the value 432.78 is assigned.
- In the above statement, the **static** modifier causes the variable to be available without an instance of its defining class being loaded and the **final** modifier makes the variable fixed.

Constant in Java

- Here a question arises that **why we use both static and final modifiers to declare a constant?**
- If we declare a variable as **static**, all the objects of the class (in which constant is defined) will be able to access the variable and can be changed its value. To overcome this problem, we use the **final** modifier with a static modifier.
- When the variable defined as **final**, the multiple instances of the same constant value will be created for every different object which is not desirable.
- When we use **static** and **final** modifiers together, the variable remains static and can be initialized once. Therefore, to declare a variable as constant, we use both static and final modifiers. It shares a common memory location for all objects of its containing class.

Operators

- Operators are symbols used to perform simple computations
  - Addition: +
  - Subtraction: -
  - Multiplication: *
  - Division: /
  - Assignment: =

- Addition, subtraction and multiplication perform as expect, but there is a minor difference with division.

Expression

- An expression is a construct made up of variables and operators that evaluates to a single value.

- Examples:
  - int sum = 20 + 40;
  - int x = a * b;
  - int inMinutes = hours*60 + minutes;
  - int z = a + c – z * 10

Order of Operations

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.

- Follows standard math rules:
  1. Parentheses
  2. Multiplication and division
  3. Addition and subtraction

- Evaluate the following expressions
  - 2 * 3 – 1
  - 2 * (3 – 1)

Order of Operations

- If the operators have the same precedence they are evaluated from left to right.

- Evaluate the following expressions
  - 10*8/2
  - 5 * 6/ 3
  - 5* (6/3)

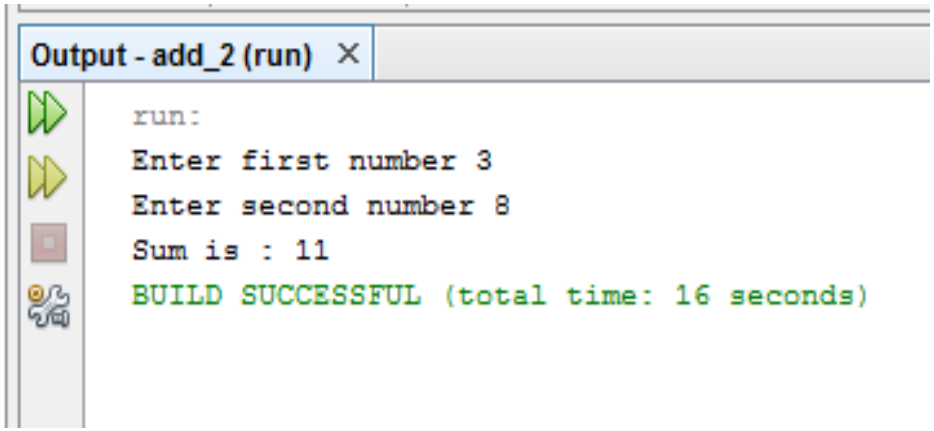
Integer Division

- Give the following ten integers below, calculate their average using a java program.

  80, 20, 100, 60, 40, 43, 3, 300, 130, 70

- Sum is 846
- Divide by 10, result is 84
- Why not 84.6?
  - sum and 10 are both integers.
  - Dividing two integers results in integer division, any fractional part of the calculation is truncated (*lost*).
  - Even when the fraction is closer to the next integer, it is lost. Is there a solution?

```
package add_2;

/**
 * @author ag
 */

import java.util.Scanner;
public class Add_2 {

public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
            int num1;
            int num2;
            int sum;

            System.out.print("Enter first number");
            num1 = input.nextInt();

            System.out.print("Enter second number");
            num2 = input.nextInt();

            sum = num1 + num2;

            System.out.printf("Sum is : %d\n", sum);
    }
}
```



Import Declaration

- Import
  - Helps to locate a class that is used in the program
  - Rich set of predefined classes that you can reuse rather than "reinventing the wheel"
  - Classes are grouped into packages- named groups of related classes- and are collectively referred to as Java Class
  - Library or the Java Application Interface (Java API)

Forgetting to include an import declaration for a class results in a syntax error.

Variable declaration

- Variable declaration statement

  Scanner input = new Scanner(System.in);

  Specifies the name(input) and type (Scanner)

  Scanner – Enables a program to read data from user

  Before using a Scanner , it must be created and specified as the source of the data.

  The equal to (=) sign is being used to initialize the variable

  The keyword ‘new’ creates an object

  System is a class, part of the Java.lang and it is not imported

- Scanner method nextInt

  num1 = input.nextInt();// read the first number from the user

- Arithmetic Operation
- Integer formatted output
  - System.out.printf(“Sum is : %d\n”, sum);
  - Format specifier %d is a placeholder for int value
  - The letter d stands for ‘decimal integer’

Operators for Strings

- In general, cannot perform mathematical operations on Strings.
  - word1 – 1
  - name/10

- The + operator works with Strings!
  - It performs concatenation.
  - i.e. joining up the two String operands by linking them end-to-end.

  - String firstName = “John”;
  - String lastName = “ Doe”;
  - System.out.println(firstName + lastName);

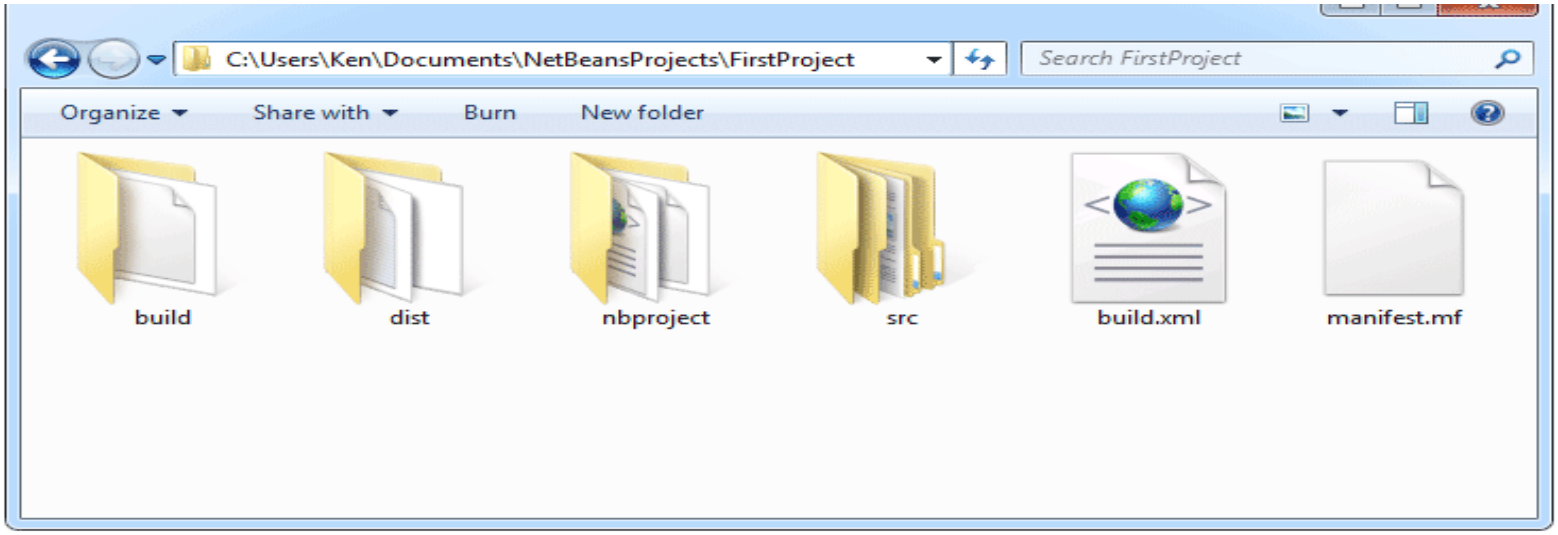
Further Exercises.

- You have to create a java program for each of example and exercise covered in the slides.
  - We have already done it for most of the examples/exercises, complete those which are missing.

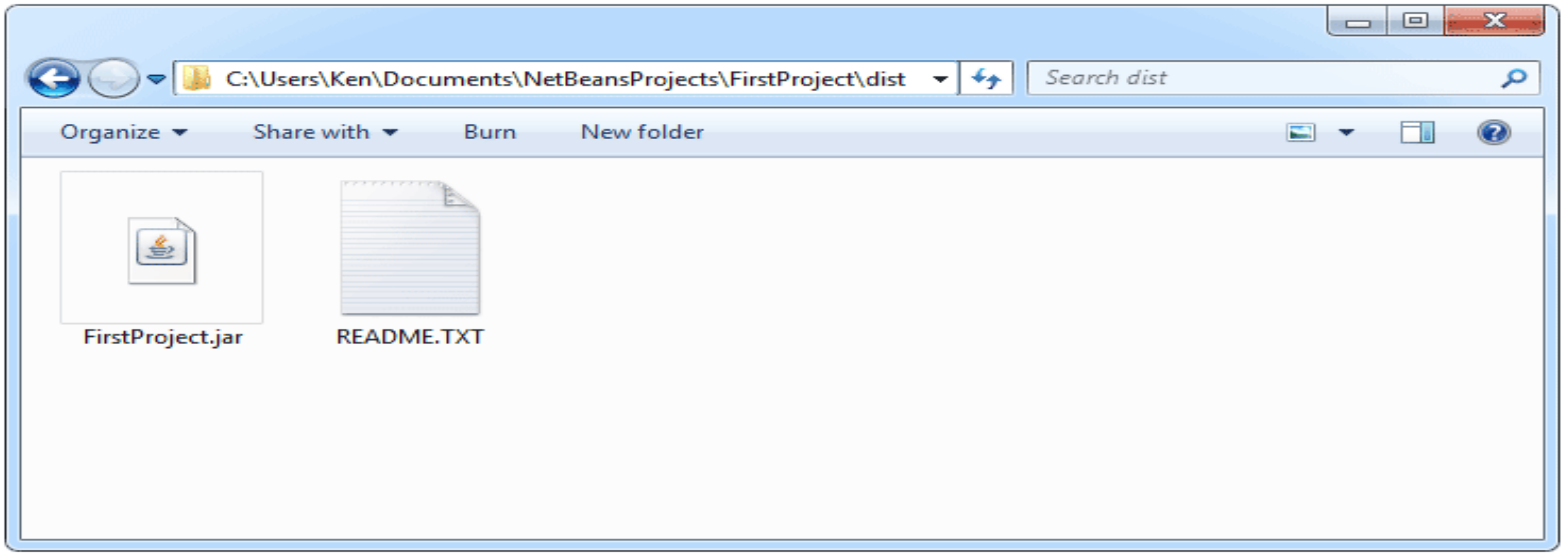
- A cyclist accelerates steadily from rest at 2 ms^{-2} for 5s. Write a java program to calculate its final velocity using the formula **v = u + at.**
  - v – final velocity, u – initial velocity, a – acceleration and t – time taken
  - u = 0, a = 2, t = 5

Sharing your java programs

- You can send your programs to other people so that they can run them. To do that, you need to create a JAR file (Java Archive). NetBeans can do all this for you. From the **Run** menu at the top, select **Clean and Build Main Project**.
- When you do, NetBeans saves your work and then creates all the necessary files. It will create a folder called **dist** and place all the files in there. Have a look in the place where your NetBeans projects are and you'll see the **dist** folder:



Double click the dist folder to see what's inside of it:



You should see a JAR file and README text file. The text file contains instructions on how to run the program from a terminal/console window.