

# SS-1201 Lab01: Getting started with Java

## Objectives

1. Learning how to Configure and use DrJava
2. Introduction to the elements of a Simple Java program:

Completed?	Sub-topics
Yes/No	Comments
Yes/No	Indentation
Yes/No	Identifiers and naming conventions
Yes/No	Keywords and Standard identifiers
Yes/No	Local variable declarations
Yes/No	Assignment and Console output methods (print, println)

3. Learning how to detect different types of errors in programs

Completed?	Sub-topics
Yes/No	Syntax errors
Yes/No	Runtime errors
Yes/No	Logic errors

4. Practise using examples
5. Complete all Lab tasks/exercises

## 1. Introduction

The main purpose of this lab is to introduce you to the computing environment of your laboratory. You will use the ideas in this lab repeatedly throughout this course, so you should make every effort to understand not only *what*, but *why* you are doing what you are doing at each step.

To be able to write, compile and execute Java applications, you need a Java Environment. Java environment includes a number of development tools, classes and methods. The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

**Java Development Kit (JDK)** – The JDK comes with a set of tools that are used for developing and running Java program. It includes:

1. Appletviewer.exe ( It is used for viewing applets)
2. Javac.exe (Java Compiler)
3. Java.exe (Java interpreter)
4. Javadoc.exe (It is for creating HTML document)
5. Jdb.exe (Java debugger)

## 2. Download and Install JDK 8

1. Go to Java Standard Edition (SE) Development Kit Download Webpage:  
<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>
2. Click on the download link that is compatible with your OS, e.g [jdk-8u261-windows-i586.exe](#) for Windows 64.
3. This will bring you to the window shown below:

You must accept the [Oracle Technology Network License Agreement for Oracle Java SE](#) to download this software.

☐ I reviewed and accept the Oracle Technology Network License Agreement for Oracle Java SE Required

You will be redirected to the login screen in order to download the file.

Download jdk-8u261-windows-i586.exe

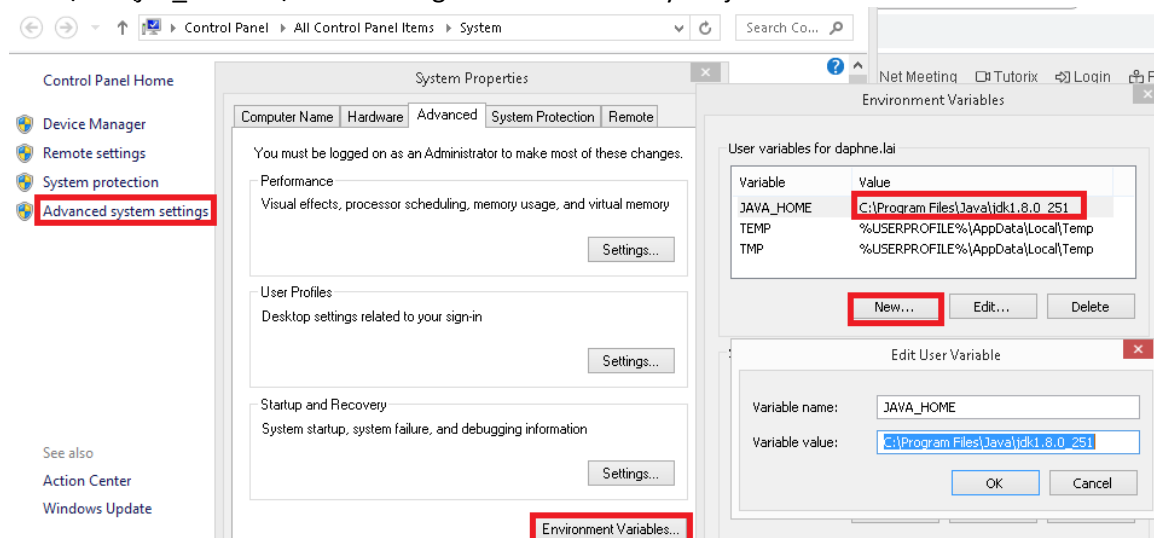
4. Please check on the reviewed and accept checkbox and click on the grey button below.
5. This will bring you to an Oracle log in page. Please sign up and this will lead you to the download of the jdk.exe installer file.
6. Once downloaded, please follow the standard wizard to complete the installation.
7. Once installation is done, check that it is done by going to command prompt (or type CMD in the search bar of Windows) and then type javac on the command prompt terminal. You should have the output below:

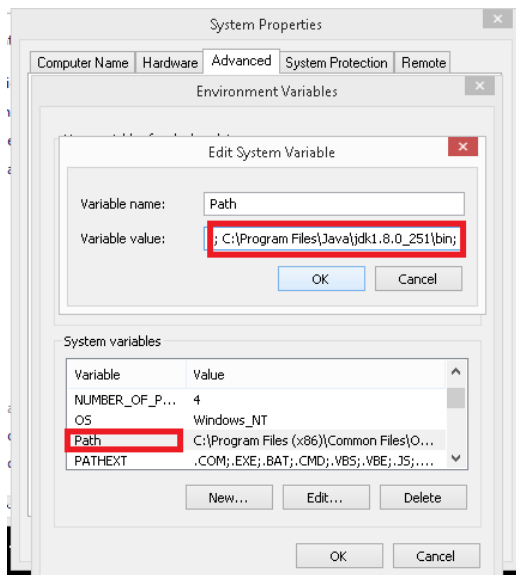
```
Command Prompt
C:\Program Files\Java\jdk1.8.0_251\bin>javac
Usage: javac <options> <source files>
where possible options include:
-g          Generate all debugging info
-g:none     Generate no debugging info
-g:{lines,vars,source}  Generate only some debugging info
-nowarn     Generate no warnings
```

8. If you get the following output, it could be that your java path has not been set.

```
C:\Program Files>javac
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

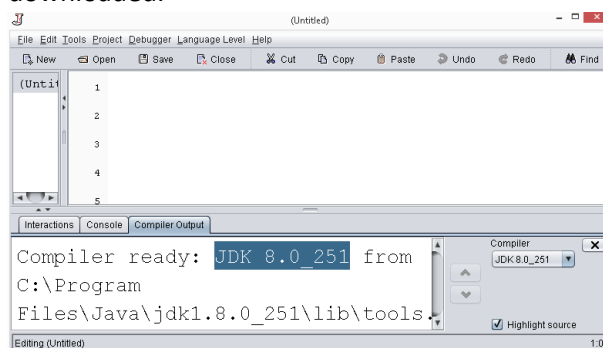
9. You may want to set java path to allow you to access java compiler from any directory in the command prompt by following the tutorial here: <https://www.tutorialspoint.com/how-to-set-temporary-and-permanent-paths-in-java>. Be sure to edit the line C:\Program Files\Java\jdk\_version\bin according to the location of your jdk folder.





### 3. Download Dr Java

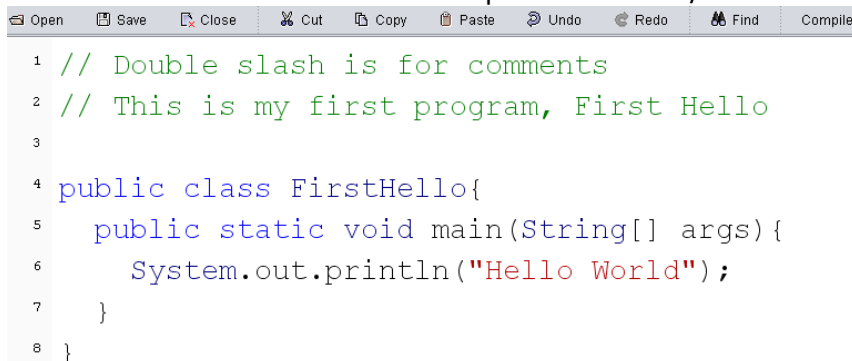
1. Download the Dr Java suitable for your OS here: <http://www.drjava.org/download.shtml>
2. Click on the downloaded file and check that the JDK version in Dr Java is the same one you downloaded.



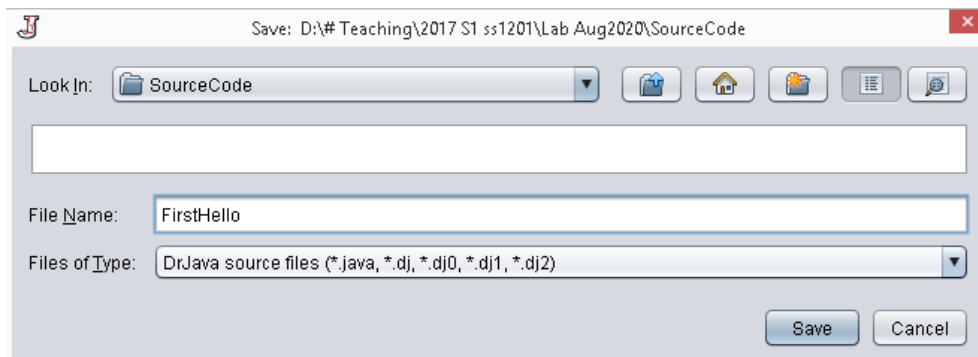
### 4. How to Create, Compile and Execute a Java program

#### Step 1: Creation or modification of a Java program :-

1. Write or modify the Java program using any text editor or IDE as shown below.
2. Notice the *class* name *FirstHello* must be the same as the filename, case sensitive.
3. Notice the difference between **scope** which ends with opening *braces* { and statements which ends with semi-colon ;
4. Braces that are opened must be closed.
5. Statements are contained within a scope of a class and/or a method.



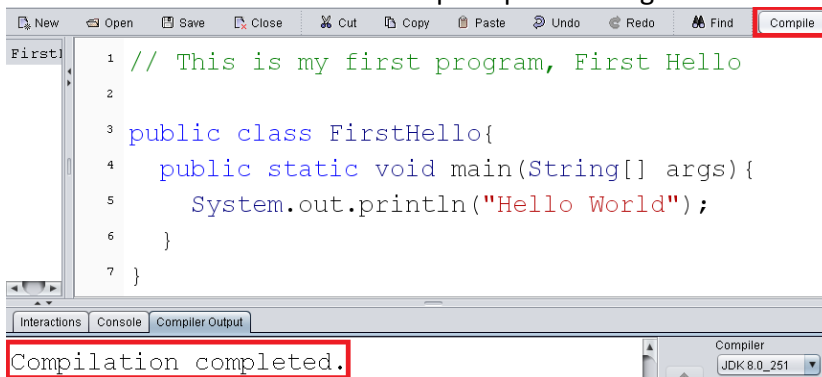
6. Save the program on secondary storage device as hard drive with a **.java** extension.



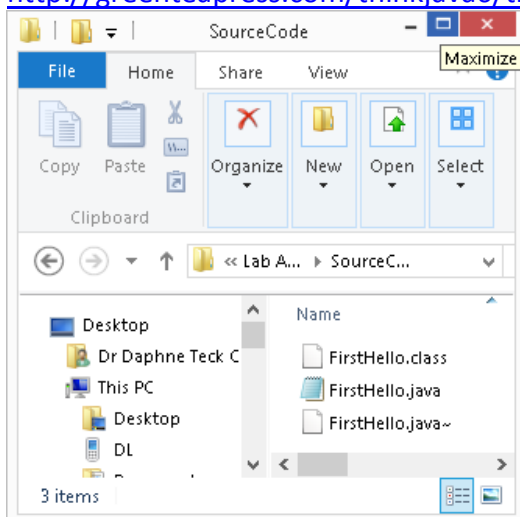
7. A java source file can contain more than one class; one of which must be declared as **public**. The name of a java source file must be the same as the name of the public class it contains.

## Step 2: Compiling a Java Program

1. Compile the program using the **javac** compiler. By a command such as:  
**javac FirstHello.java**
2. This can be done at the command prompt or through an IDE.



3. After executing the javac command, if there are no syntax errors (i.e., grammatical errors) in our program we get a **.class** file which contains an intermediate machine language code called the java bytecode. Please read Chapter 1 of Think Java book:  
<http://greenteapress.com/thinkjava6/thinkjava.pdf>



4. If a source file has more than one class, each class is compiled into a separate class file.

## Step 3a: Running (Executing) a Java Program

1. To execute **.class** files they must be translated to the native machine language; this is done by invoking the Java Virtual Machine (JVM) by using the **java** interpreter.

The screenshot shows an IDE window with a menu bar (New, Open, Save, Close, Cut, Copy, Paste, Undo, Redo, Find, Compile, Reset, Run) and a toolbar. The main editor displays a Java program named 'FirstHello' with the following code:

```
1 // This is my first program, First Hello
2
3 public class FirstHello{
4     public static void main(String[] args){
5         System.out.println("Hello World");
6     }
7 }
```

Below the editor, there are three tabs: 'Interactions', 'Console', and 'Compiler Output'. The 'Interactions' tab is active and shows the command prompt output:

```
> run FirstHello
Hello World
```

2. The command is similar to going to the directory of your source code in the first box and typing as shown in the second box:

**java FirstHello**

The screenshot shows a command prompt window with the following text:

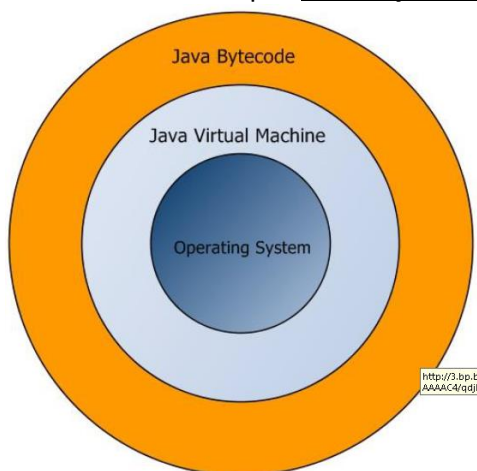
```
D:\# Teaching\2017 S1 ss1201\Lab Aug2020\SourceCode>java FirstHello
Hello World
```

This can be done at the command prompt (above, shown in 2) or using an IDE (shown in 1).

3. We can see the output in the third red box "Hello World".

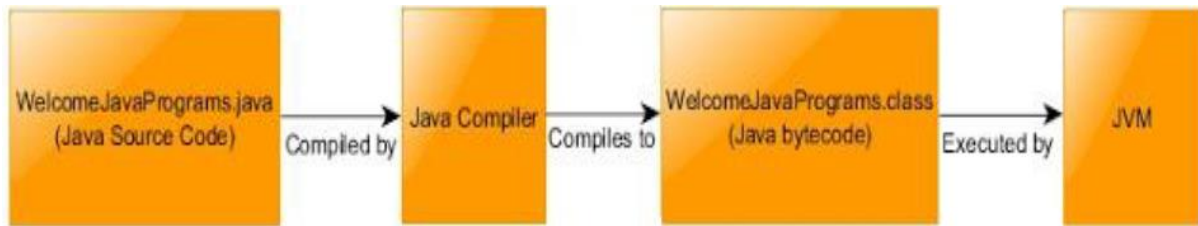
### Step 3b: Program loading into memory by JVM:

JVM loads the **.class** file to primary memory before its execution. The JVM does this by using a class loader. All the **.class** files required by our program for the execution are loaded by the class loader into the memory just before the execution. Please go through Lecture Slides and Think Java Book on the topic: **The way of the program** for better understanding.



### Step 3c: Bytecode Verification by JVM :-

In order to maintain security of the program JVM has bytecode verifier. After the classes are loaded in to memory, the bytecode verifier verifies the bytecode of the loaded class or classes in order to maintain security. It checks whether the bytecodes are valid. Thus, it prevents our computer from malicious viruses and worms.



### Step 3d: Execution of Java program: -

Whatever actions we have written in our Java program, JVM executes them by interpreting bytecode.

## 3. The elements of a simple Java program

### 3.1. General Structure

```

public class ClassName{
    public static void main(String[] args){
        statement1;
        statement2;
        // ...
        statementN;
    }
}
  
```

### 3.2 Comments

**Comments** are English sentences inserted in a program to explain its purpose; they are ignored by the compiler.

We use comments to explain the purpose of a class, a method or a variable.

1. **Single-Line comments** begin with two forward slashes. The compiler ignores everything from the two forward slashes to the end of the line. `// ...`
2. **Multi-Line comments** start with a forward slash followed by an asterisk, `/*`, and are terminated by an asterisk followed by a forward slash, `*/`. The compiler ignores everything between the `/*` and the `*/`. This type of comment can span multiple lines. `/* ... */`

**Note:** You cannot nest a comment `/* .... */` inside another comment `/* .... */`

### 3.3 Primitive data types

All data in Java falls into one of two categories: **primitive data** and **objects**.

- A primitive data value uses a small, fixed number of bytes.
- There are only eight primitive data types (Example: int, double).
- A programmer cannot create new primitive data types.

### 3.4 Keywords and Identifiers

- An identifier is the name of an item in a program (e.g., variables, named constants, method names, class names etc.).
- **Java is a free form, case-sensitive language.**
- Identifier naming rules in Java are:
  1. The first character of an identifier must be a letter, underscore, or \$
  2. All the other characters of an identifier must be letters, digits, or underscores
  3. An identifier may be arbitrarily long (A compiler may impose a limit on the length of an identifier)

4. The name of an identifier cannot be one of the words that the Java language has reserved for its own use. A reserved word is also called a keyword. This means that you cannot use one of the Java keywords to name your identifiers.
  5. Some keywords: **int, double, float, new, void, if, else, switch, do, while, for**
- Java has some pre-defined words or **Standard Identifiers** that are not keywords. You should not use these pre-defined words to name your identifiers to avoid confusion. Examples are names of standard classes and names of standard methods: **System, Math, Scanner, String, print, println, printf** etc.

### 3.5 Declaring and Using Variables.

A variable declaration is a request to the system to allocate memory for data values to be stored. e.g.

```
int age;  
int numberOfStudents = 20;  
double average, sum;
```

### 3.6 The scope of a local variable

The scope of a variable is the part of the program over which the variable name can be used. You cannot use a variable before its declaration; otherwise the compiler issues a compilation error: **“error: cannot find symbol”**

In Java, matching curly brackets { } define a block. A variable that is declared within a method block is called a local variable.

The scope of a local variable is from the point it is declared to the end of the block in which it is declared, including nested blocks that appear after the declaration. Statements outside this region cannot access such a local variable.

A local variable must be initialized (i.e., it must have a value) before it can be used in an expression; otherwise the compiler issues a compilation error: **“error: variable might not have been initialized”**

You cannot declare two variables with the same name in the same scope; otherwise the compiler issues a compilation error: **“error: variable already defined in method”**

### 3.7 Arithmetic Operators and Arithmetic Expressions

Arithmetic operators (+, -, \*, %), and operands are used to form arithmetic expressions.

### 3.8 Output Statements

- In programming, a statement is an instruction to a computer to do something.
- A statement forms part of the sequence of program execution.
- **In Java, every simple statement is terminated with a semicolon.** Example:  
`System.out.println("SS 1201 Lab01");`
- Multiple simple statements can be written on a single line. Example:  
`System.out.println("SS 1201 Lab01"); System.out.println("Java");`

### 3.9 Assignment Statements

We can change the value of a variable any number of times within a program. One-way of assigning or changing the value of a variable is using the assignment statement, which has the form: *variable = expression*;

Example:

```
int first, second, count = 0;
double average;
first = 10;
second = 20;
average = (first + second)/2.0;
count = count + 1;
```

#### Multiple assignments in a single statement

Java supports multiple assignments in a single statement. Example:

```
int a, b, c; a = b = c = 250;
```

Note: The code below will produce a compilation error: **“error: cannot find symbol”**

```
int a = b = c = 250;
```

You cannot use multiple assignments in a declaration statement.

### 3.10 String concatenation operator

The operator + when applied to two strings creates a new String object that is the concatenation of the two string objects. The operator may also be applied to a String and a numeric expression to produce a String object that concatenates the String and the expression. The String concatenation operator has the same priority as the addition and subtraction operators.

Example: The code:

```
String mystring = "UBD";
System.out.println("I am a student of " + mystring);
int num1 = 12, num2 = 45;
System.out.println(num1 + " + " + num2 + " = " + (num1 + num2));
String mystring = "UBD";
System.out.println("I am a student of " + mystring);
int num1 = 12, num2 = 45;
System.out.println(num1 + " + " + num2 + " = " + (num1 + num2));
```

produces the output:

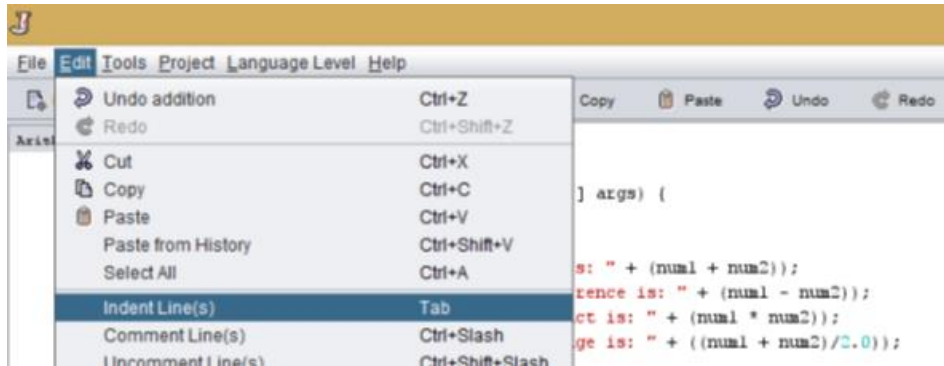
```
I am a student of UBD
12 + 45 = 57
```

### 3.11 Indentation

Indentation involves using *spaces*, and *blank lines* to visually group related statements together.



Note: In DrJava, the menu **Edit** → **Indent Line(s)** can be used to format your program:



## 4. Types of Program Errors

Program errors are also referred to as program bugs.

A Java program may have one or more of three types of errors:

- Syntax errors (Compiler errors or Compile-time errors)
- Runtime errors
- Logic errors

Usually, the errors become more difficult to find and fix as you move down the above list.

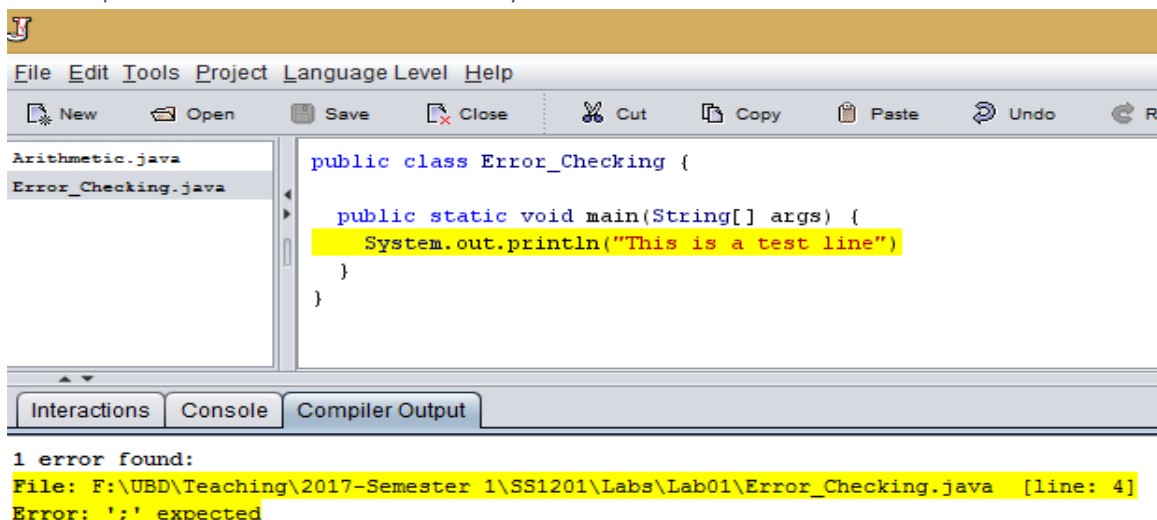
### 4.1 Syntax errors

Syntax errors represent *grammar errors* in the use of the programming language. Common examples are:

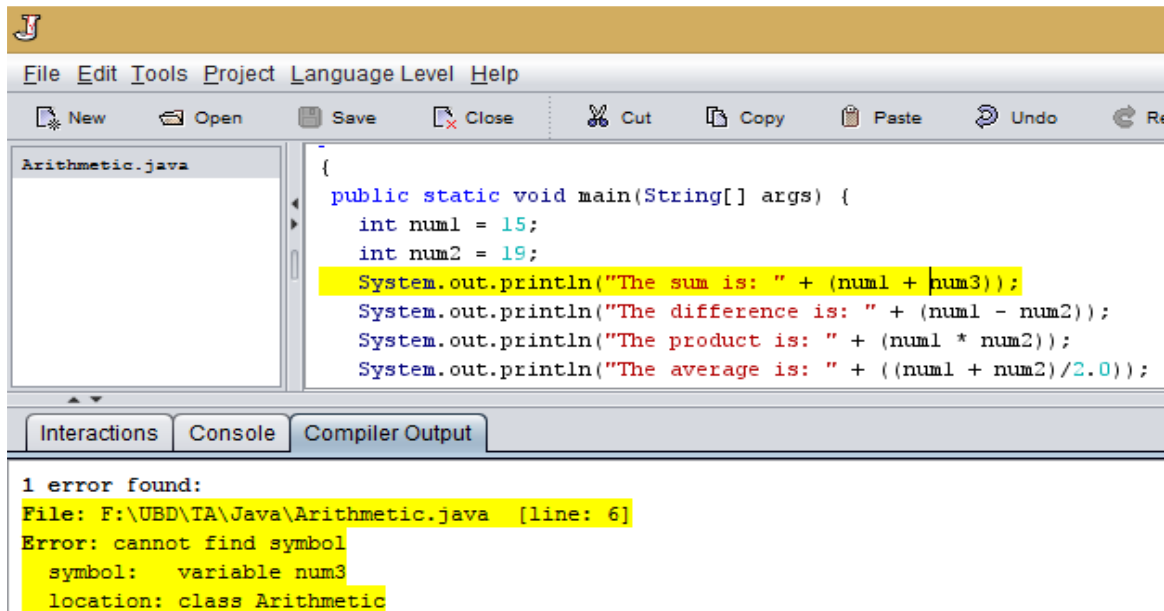
- a. Misspelled variable and method names
- b. Missing semicolons
- c. Unmatched parentheses, square brackets, and curly braces
- d. Unclosed String constants
- e. Using a variable that has not been declared
- f. Incorrect format in selection and loop statements

Syntax errors are the easiest to find and fix. Over the years, compiler developers have worked hard to make compilers smarter so that they can catch errors at compile time that might otherwise turn out to be runtime errors. Examples of Syntax errors:

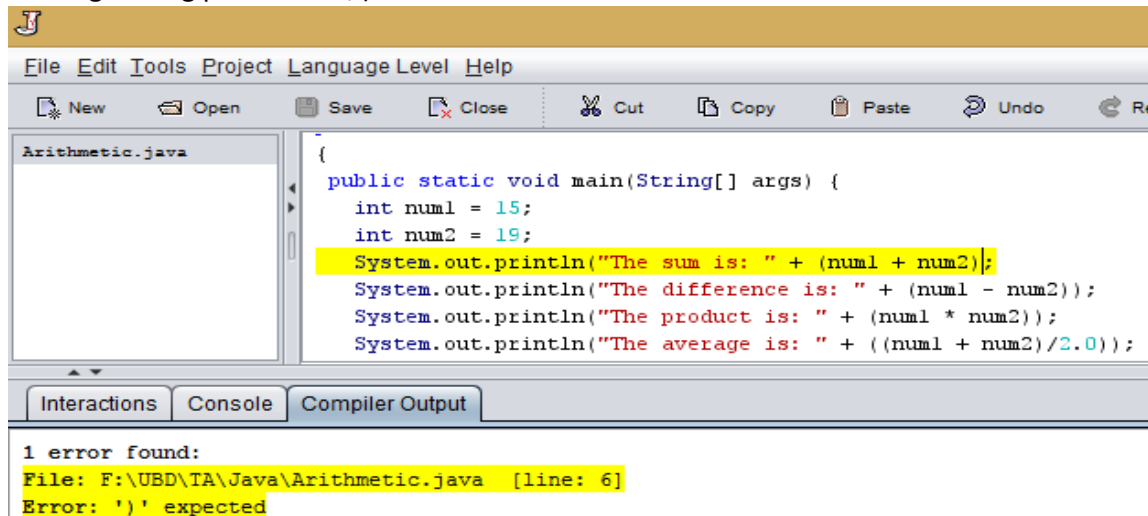
1. Simple statement not terminated by semicolon:



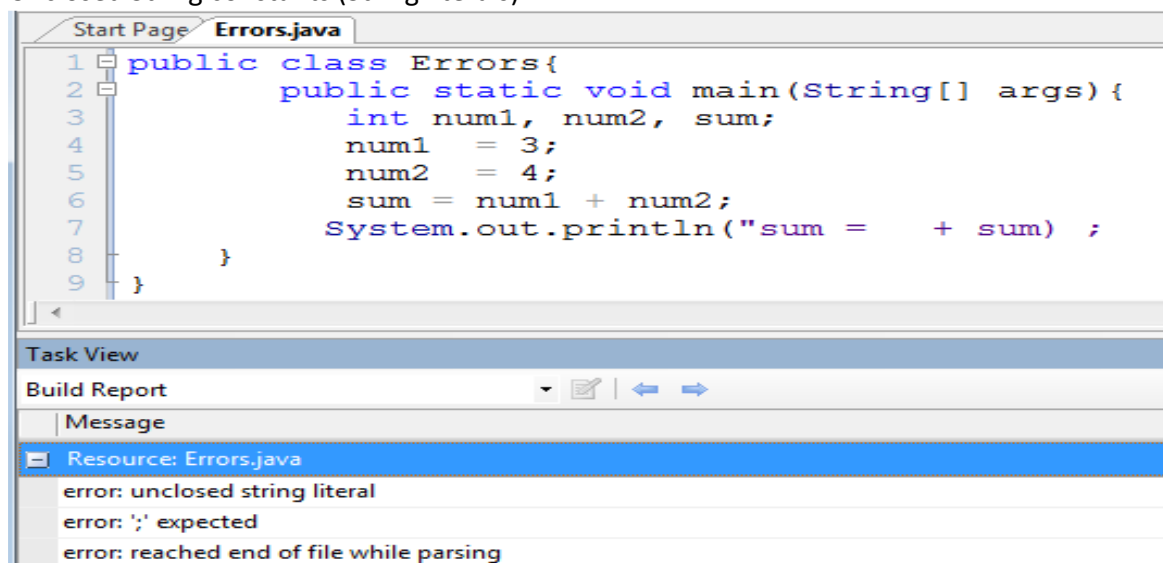
2. Using a variable before it is declared:



3. Missing closing parenthesis, ) :



4. Unclosed String constants (String literals):



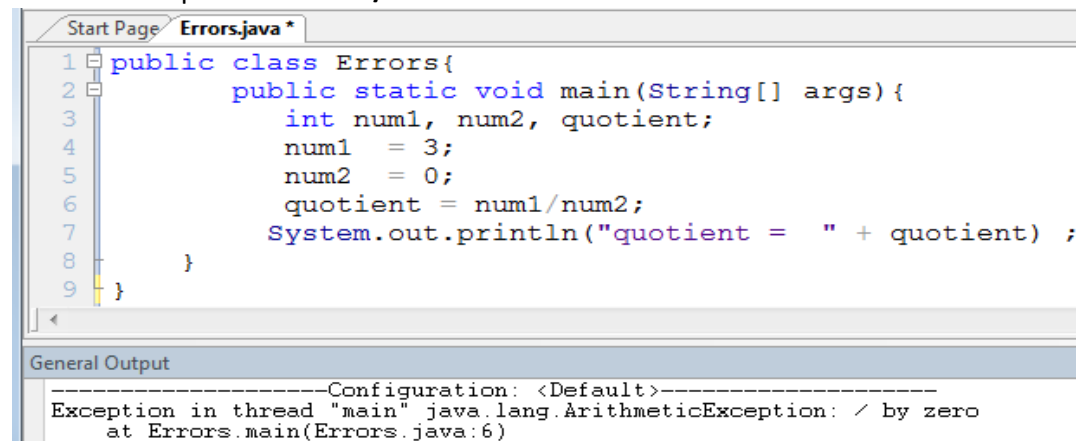
## 4.2 Runtime errors

A type of error that occurs during the execution of a program is known as run-time error. Runtime errors may crash your program when you run it. Runtime errors occur when a program with no syntax errors directs the computer to execute an illegal operation. Common examples are:

- Trying to perform integer division by a variable that contains a value of zero
- Trying to open a file that does not exist

There is no way for the compiler to know about these kinds of errors when the program is compiled. Runtime errors are commonly due to wrong input from the user. Runtime errors are usually more difficult to find and fix than syntax errors.

To find the source of a run-time error in a program, usually a software called **debugger** is used. Example: When the following program is executed, a run-time error occurs due to division by zero in the expression **num1 / num2**



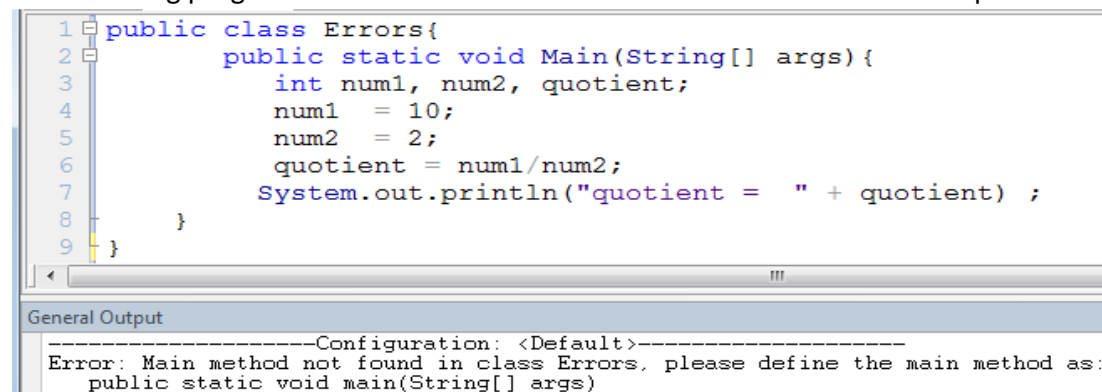
The screenshot shows an IDE window titled "Errors.java \*". The code is as follows:

```
1 public class Errors{
2     public static void main(String[] args){
3         int num1, num2, quotient;
4         num1 = 3;
5         num2 = 0;
6         quotient = num1/num2;
7         System.out.println("quotient = " + quotient) ;
8     }
9 }
```

Below the code editor is the "General Output" pane. It displays the following message:

```
-----Configuration: <Default>-----
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Errors.main(Errors.java:6)
```

The following program has a run-time error because the main method is misspelled:



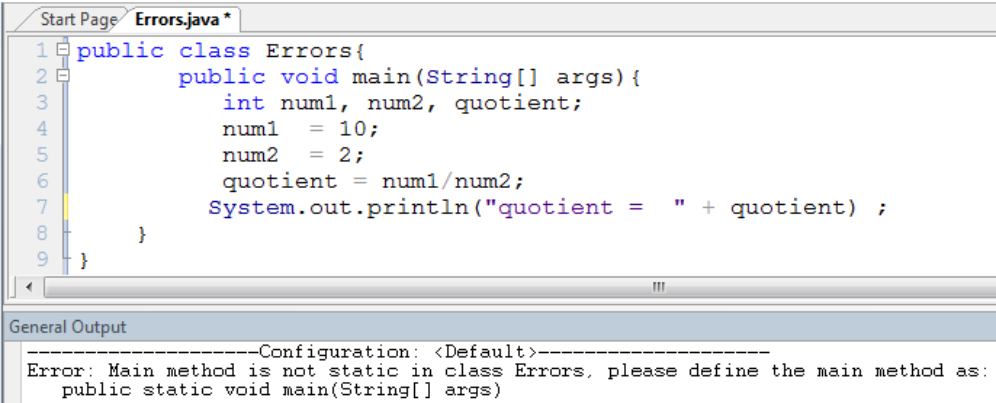
The screenshot shows an IDE window with the following code:

```
1 public class Errors{
2     public static void Main(String[] args){
3         int num1, num2, quotient;
4         num1 = 10;
5         num2 = 2;
6         quotient = num1/num2;
7         System.out.println("quotient = " + quotient) ;
8     }
9 }
```

The "General Output" pane shows the following error message:

```
-----Configuration: <Default>-----
Error: Main method not found in class Errors, please define the main method as:
public static void main(String[] args)
```

The following program has run-time error because the main method is not declared as static:



### 4.3 Logic errors

Logic errors occur when a programmer implements the algorithm for solving a problem incorrectly. A statement with logical error may produce unexpected and wrong results in the program. Common examples are:

- Multiplying when you should be dividing
- Adding when you should be subtracting
- Opening and using data from the wrong file
- Displaying the wrong message

Logic errors are the hardest to find and fix because:

- The compiler does not detect these errors
- There is no indication of error when the program is executed.
- The program may produce correct results for some input data and wrong results for other input data.

Logic errors can only be detected by examining the program thoroughly. This is usually done by using a **debugger**.

Example: The following program has a logic error in the **System.out.println()** statement:

```

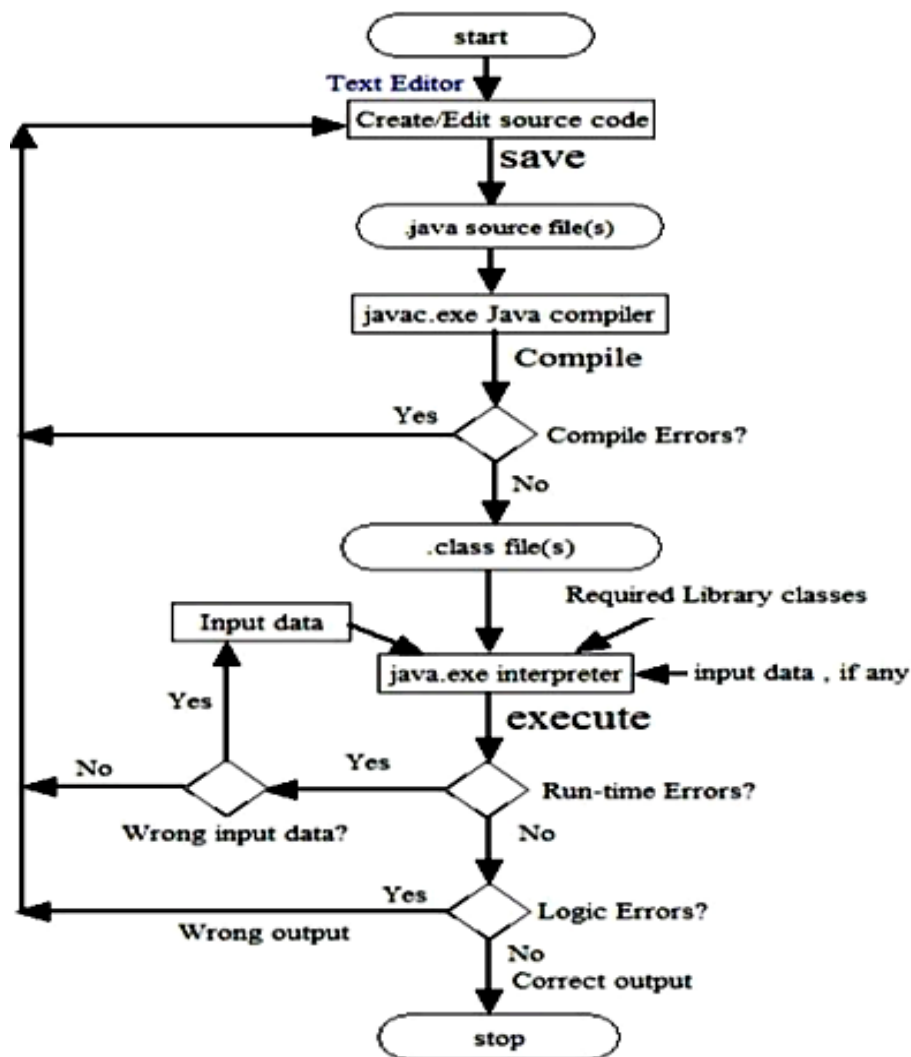
public class Errors{
    public static void main(String[] args){
        int num1 = 4, num2 = 9;
        System.out.println(num1 + " + " + num2 + " = " + (num1 * num2)); }
}

```

Notice that the program produces correct results for some initializations (num1 = 2, num2 = 2 and num1 = 0, num2 = 0) and wrong results for others:

2 + 2 = 4	3 + 4 = 12
-----------	------------

Since a Java program may have syntax, logic, or run-time errors, the edit-compile-execute cycle consists of the steps similar to those depicted in the diagram below:



## 5. Examples

**Example 1:** The following program assigns two values to num1 and num2 and prints the following:

- Their sum
- Their difference
- Their product
- Their average

```

/* computes the sum, difference, product of two integer numbers */
public class Arithmetic {
    public static void main(String[] args) {
        int num1 = 15;
        int num2 = 19;
        System.out.println("The sum is: " + (num1 + num2));
        System.out.println("The difference is: " + (num1 - num2));
        System.out.println("The product is: " + (num1 * num2));
        System.out.println("The average is: " + ((num1 + num2)/2.0));
    }
}

```

**Note:** By introducing additional variables, the previous program may be written as:

```

/* computes the sum, difference, product of two integer numbers */
public class Arithmetic {
    public static void main(String[] args) {
        int num1 = 15 , num2 = 19, sum, diff, product;
        double average;
        sum = num1 + num2;
        diff = num1 - num2;
        product = num1 * num2;
        average = (num1 + num2) / 2.0;
        System.out.println("The sum is: " + sum);
        System.out.println("The difference is: " + diff);
        System.out.println("The product is: " + product);
        System.out.println("The average is: " + average);
    }
}

```

**Example 2:** The following program shows the difference between `System.out.print()` and `System.out.println()`. Pay attention to the spaces. Try and run: What happens when you remove space in the first print statement?

```

class Hello {
    public static void main(String[] args) {
        System.out.print("Goodbye, ");
        System.out.println("cruel world!");
    }
}

```

**Example 3:** Before typing this out on the editor, what do you think will be the output?

```

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

```

**Example 4:** In this example, we practise on the difference between declaration and assignment; also we note the difference between print and println. Try: What happens if bob is assign the value of 3? Explain your observation.

```

class BobHourMin {
    public static void main(String[] args) {
        String bob; //declaration
        int hour, minute; //declaration

        bob = "Hello."; // assign bob the value "Hello."
        hour = 11; // assign the value 11
        minute = 59; // assign minute to 59

        System.out.println("bob has the value of " + bob);
        System.out.print("hour has the value of ");
        System.out.println(hour);
        System.out.println("minute has the value of " + minute);
    }
}

```

**Example 5:** Understanding the data types passed in the println arguments.

```

public class Test {
public static void main(String[] args){
    System.out.println("39 + 3");
    System.out.println(39 + 3);
}
}

```

**Example 6:** Understand the issues with dividing by integer and double. First try the code below and study carefully the calculations according to the values stored in the variable. Next modify in HourMin and save the file as HourMinTwo, the last statement 60 to be changed to the value 60.0.

What did you learn?

```

public class HourMin {
    public static void main(String[] args){
        int hour, minute;
        hour = 11;
        minute = 59;
        System.out.print("Number of minutes since midnight: ");
        System.out.println(hour*60 + minute);
        System.out.print("Fraction of the hour that has passed: ");
        System.out.println(minute/60);
    }
}

```

**Example 7:** Understand the order of assignments affect the values stored in the variables. Why is the variable **temp** required?

```

public class SwapXY {
    public static void main(String[] args){
        double x = 20.5, y = -16.7, temp;
        System.out.println("Before: x= "+x + "; y= " + y);
        temp = x;
        x = y;
        y = temp;
        System.out.println("After: x= "+x + "; y= " + y);
    }
}

```

**Example 8:** Calculate the total volume of liquids from differently sized containers. How would you modify your code if the a) number of cans is now 8 cans b) the amount per can is 0.4 litres

```

1  /**
2   * This program computes the volume (in liters) of a six-pack of soda
3   * cans and the total volume of a six-pack and a two-liter bottle.
4   */
5  public class Volume1
6  {
7      public static void main(String[] args)
8      {
9          int cansPerPack = 6;
10         final double CAN_VOLUME = 0.355; // Liters in a 12-ounce can
11         double totalVolume = cansPerPack * CAN_VOLUME;
12
13         System.out.print("A six-pack of 12-ounce cans contains ");
14         System.out.print(totalVolume);
15         System.out.println(" liters.");
16
17         final double BOTTLE_VOLUME = 2; // Two-liter bottle
18
19         totalVolume = totalVolume + BOTTLE_VOLUME;
20
21         System.out.print("A six-pack and a two-liter bottle contain ");
22         System.out.print(totalVolume);
23         System.out.println(" liters.");
24     }
25 }

```

## 6. Lab Tasks /Exercises

**Each task/exercise is worth 10 marks.**

**Task 1:** Try out all the examples given above in 3.8, 3.9 and 3.10, printing any variables using `System.out.println()` where required. Include your own comments for each statement to demonstrate your understanding. Save each section in a separate java file.

**Task 2:** Repeat the errors shown in Section 4. Save codes for each type of errors in a separate java file.

**Task 3:** Try out all the examples in Section 5. Include comments to explain what each statement is doing. Save each example in a separate java file.

**Task 4:** Using the [Java API Documentation](#) document find and write down the headers of the methods:

- `sqrt`, `pow`, and `max` of the **Math** class
- `nextInt`, `nextDouble`, `nextLine` of the **Scanner** class

**Note:** If a method has more than one header, choose any one.

An example of a method header is: **`static double abs(double a)`**

**Task 5:** Read section 4. Types of Program Errors

Type the program below in **DrJava IDE**.

Save the program as **Errors.java**.

Determine, classify and correct all errors in the program by commenting on them, compiling and executing the program:



```

public Errors{
    public static void MAIN(string[] args){
        int num1, num2;
        num3 = 25;
        num2 = 16
        system.out.println(num1 + " + " + num2 + " = " + (num1 - num2));
    }
}

```

**Task 6:** The following Java program is intended to compute the average of three integers. The program has a logic error, it may not produce the correct output for some values of x1, x2, and x3 [Example, any sum of x1, x2, and x3 that is not divisible by 3 produces wrong output]. The program also does not follow the objectives of this lab (i.e., using comments, using indentation, and using meaningful identifiers).

Type the program in DrJava IDE then enhance it so that it follows Java naming conventions, proper program layout, and it does not contain the logic error that it has:

```

public class X{
    public static void main(String[] args) {
        int x1 = 1; x2 = 2, x3 = 4; int x4 = (x1 + x2 + x3) / 3;
        System.out.println(x4);
    }
}

```

**Task 7:** Design then implement a Java program that displays the numbers **1** to **4** on the same line, with each pair of adjacent numbers separated by one space, i.e., the output of the program must be: **1 2 3 4**

Write the program using **each** of the following techniques:

- Use only one **System.out.println** statement
- Use four **System.out.print** statements

**Task 8:** Design then implement a Java program to draw a square-like shape that has 4 sides, each composed of 6 asterisks (\*) in length:

```

*****
*      *
*      *
*      *
*      *
*****

```

**Task 9:** Create a Java program with the following output:

```

Hello World.
This is my first program.
Practice is the key to increasing good skills.

```

**Task 10:** Create a HelloPrinter program to print the word “Hello” vertically?

**Task 11:** Write a program that prints the sum of the first ten positive integers,  $1 + 2 + \dots + 10$ .

**Task 12:** Write a program that prints the product of the first ten positive integers,  $1 \times 2 \times \dots \times 10$ . (Use \* to indicate multiplication in Java.)

**\*Task 13:** Create a program to calculate and print the results using the formula given below. Remember to initialise (declare and assign) values for variables **x** and **y**. Explain what do Math.PI, Math.max and Math.abs do.

`Math.PI * Math.max(4 * y, Math.abs(x - y))`

**\*Task 14:** Another three two-litre bottles have been added to the group of cans and bottle in Example 8. Modify the code, rename the file as NewVolume1.java, to calculate ***newtotalvolume***.

**\*Task 15:** Write a program that prints the balance of an account after the first, second, and third year. The account has an initial balance of \$1,000 and earns 5 percent interest per year. (Hint: Calculate first year interest. The first year balance will be the sum of initial balance and first year interest. Then continue to calculate second year balance using first year balance and so on.)

\* Challenging