Programming Fundamentals
Week 1 Talk b - Wednesday

Introduction

Produced by: Siobhan Roche
Mairead Meagher

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
setu.ie

Points from Monday

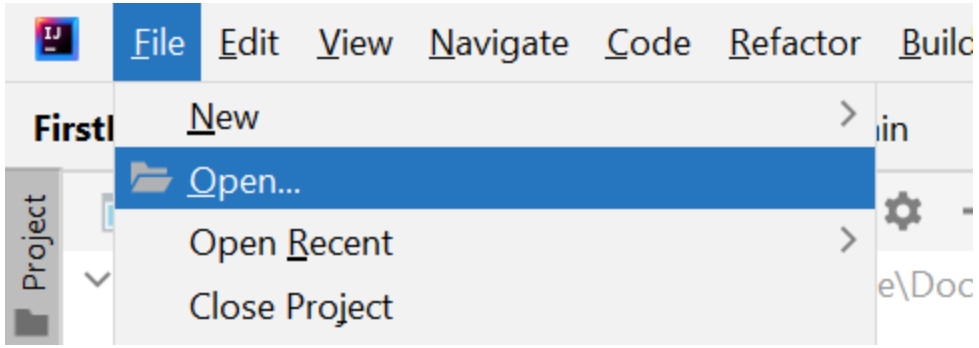
- File Name and Class name should be the same

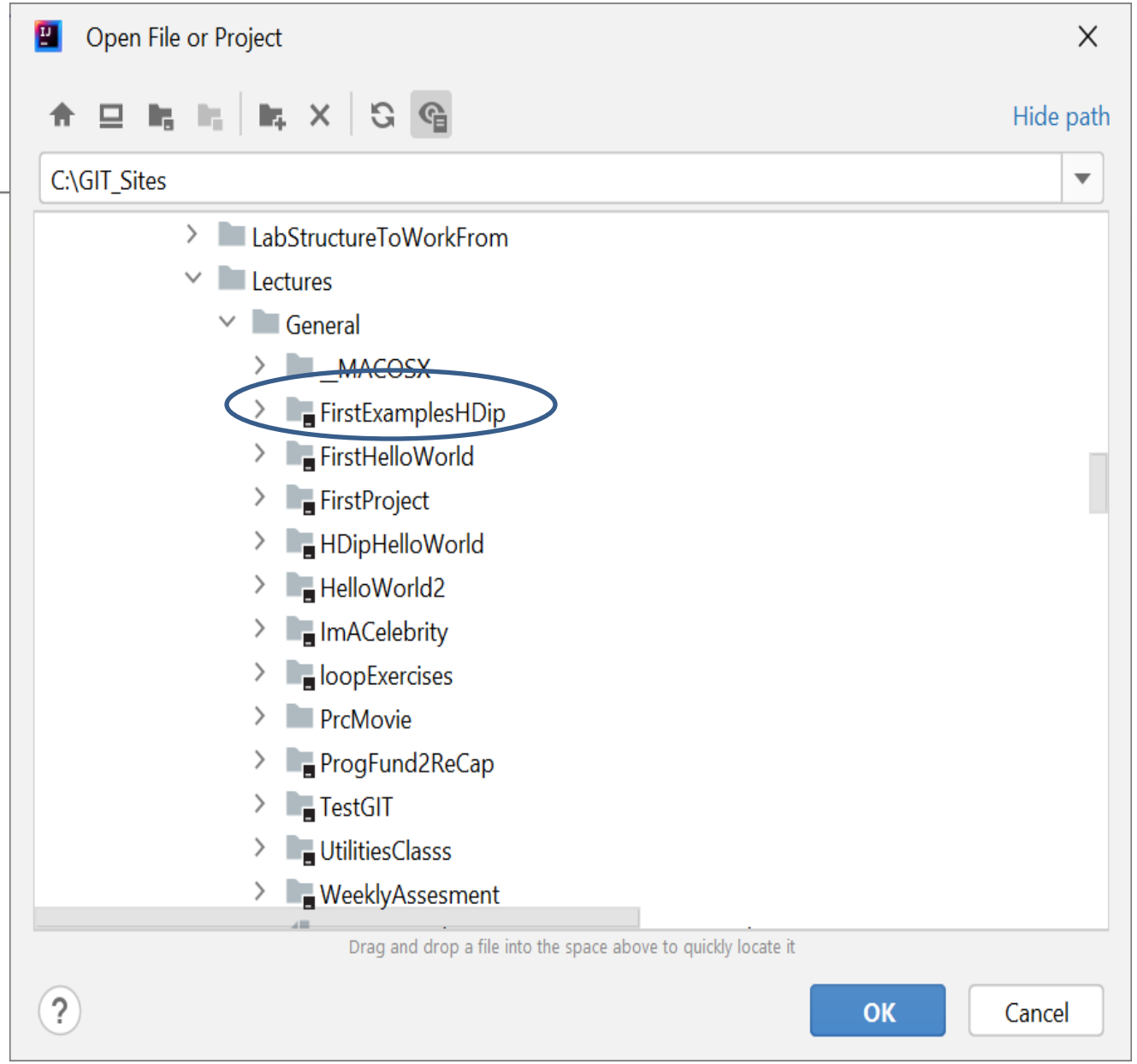
SameName.java

```
1  public class SameName {
2      public static void main(String[] args) {
3
4          System.out.println("Hello world, I'm here!");
5      }
6  }
```

Points from Monday

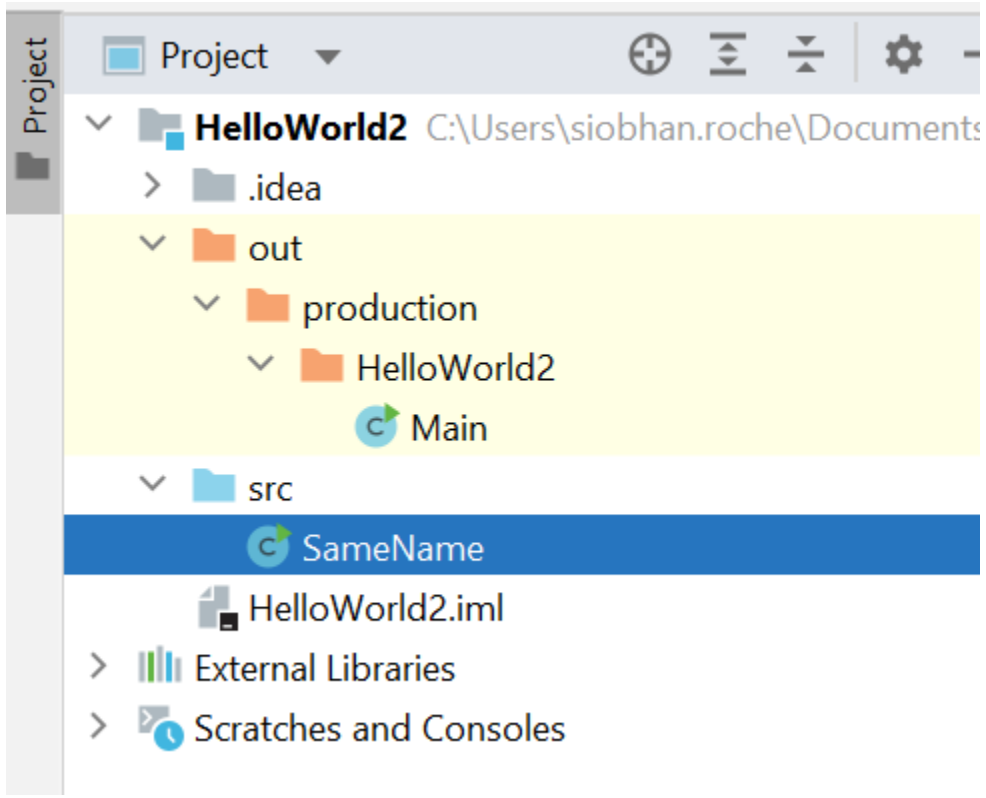
- Always open your project, not individual files





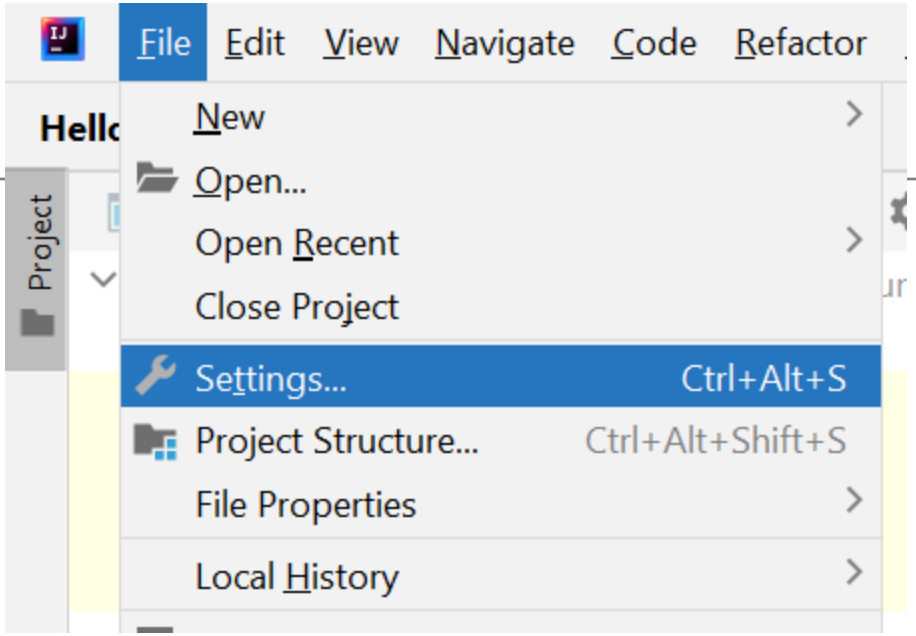
Points from Monday

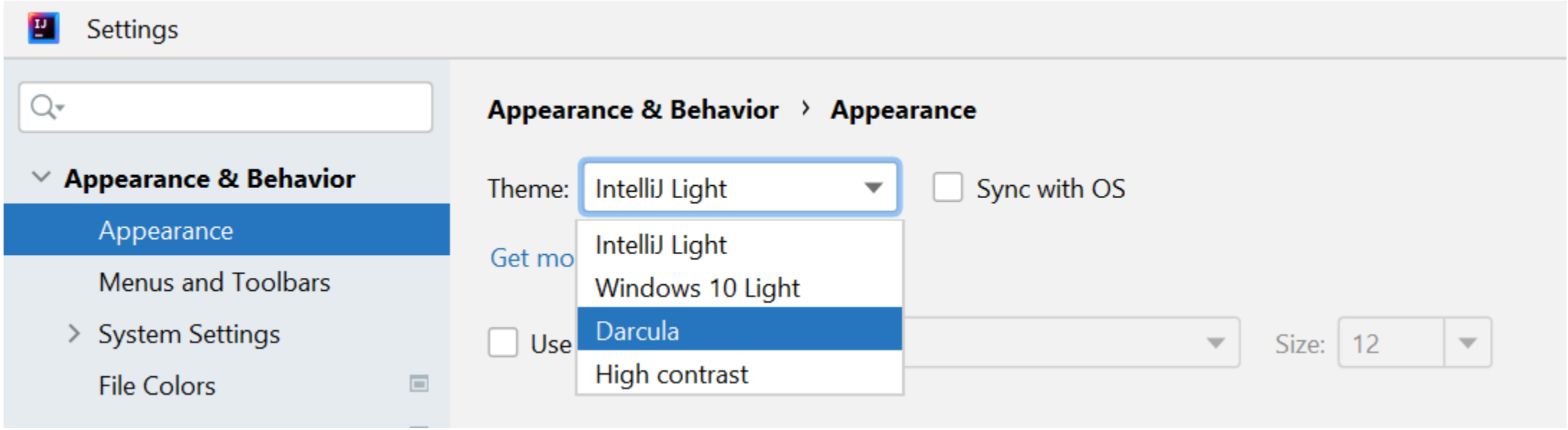
- Make sure you are in the src folder



Points from Monday

- How to change the appearance

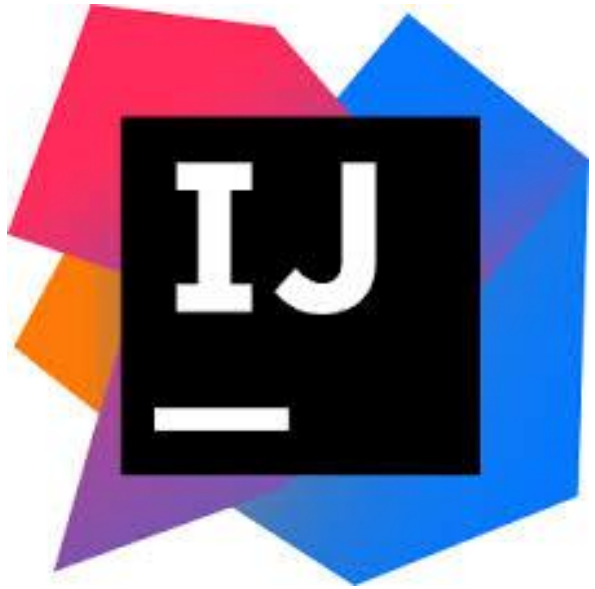




Today

- Data Variables
- Comments
- Errors

- Scanner (next week)



An Introduction to Processing

Variables, Data Types & Arithmetic Operators

Produced by:
Mr. Colm Dunphy
Dr Siobhan Drohan
Mairead Meagher
Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
setu.ie

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Variables

In Programming, **variables**:

- are created (defined) in your programs.
- are used to store data (whose value can change over time).
- have a data type.
- have a name.
- are a VERY important programming concept.

Variable **names**…

- Are case-sensitive.
- Begin with either:
  - a **letter (preferable)**,
  - the dollar sign "$", or
  - the underscore character "_".
- Can contain letters, digits, dollar signs, or underscore characters.
- Can be any length you choose.
- Must not be a **keyword or reserved word**
  - e.g. int, while, etc.
- Cannot contain white spaces.

camelCase

- capitalize each word except the first

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Variable names should be carefully chosen

- Use full words instead of cryptic abbreviations e.g.
  - variables named speed and gear are much more intuitive than abbreviated versions, such as s and g.

- If the name consists of:
  - only one word,
    - spell that word in all lowercase letters e.g. ratio.
  - more than one word,
    - capitalise the first letter of each subsequent word e.g. gearRatio and currentGear.
    - This is called **camelCase**

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Assignment Statement

- Values are stored in variables via *assignment statements*:

Syntax	`variable = expression;`
Example	`diameter = 100;`

- A variable stores a single value, so any previous value is lost.
- Assignment statements work by
  - taking the value of what appears on the right-hand side of the operator
  - and copying that value into a variable on the left-hand side.

Topics list

1. Variables.

2. Assignment statement.

3. Data Types.

4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.

5. Arithmetic operators.

Data Types

- In Java, when we define a variable, we **<u>have</u>** to give it a data type.

- The data type defines the **<u>kinds of values</u>** (data) that can be stored in the variable
  e.g.
    - - 456
    - 2
    - 45.7897
    - I Love Programming
    - S
    - true

- The data type also determines the **<u>operations</u>** that may be performed on it.

Data Types

- Java uses two kinds of data types:
  - **Primitive** types
  - **Object** types

- We are only looking at **Primitive** types now;
  we will cover Object types later in the module.

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Java's Primitive Data Types

- Java programming language supports eight primitive data types.
- A primitive type is predefined by the language and is named by a reserved keyword.
- A primitive type is highlighted red when it is typed into the PDE e.g.

```
int numberOfItems;
boolean bounceUp;
float lengthOfRectangle;
```

Java Keywords

- Words with a special meaning in the language, e.g:
  - **`public`**
  - **`class`**
  - **`private`**
  - **`int`**
- Also known as *reserved words*.
- Special words that the Java language keeps for itself.

Keywords – list

Java Keywords - for the moment, only the blues are important							
_	catch	double	float	int	private	super	TRUE
abstract	char	else	for	interface	protected	switch	try
assert	class	enum	goto	long	public	synchronized	void
boolean	const	extends	if	native	return	this	volatile
break	continue	FALSE	implements	new	short	throw	while
byte	default	final	import	null	static	throws	
case	do	finally	instanceof	package	strictfp	transient	

Keywords (technical description)

A token is small meaningful piece of code

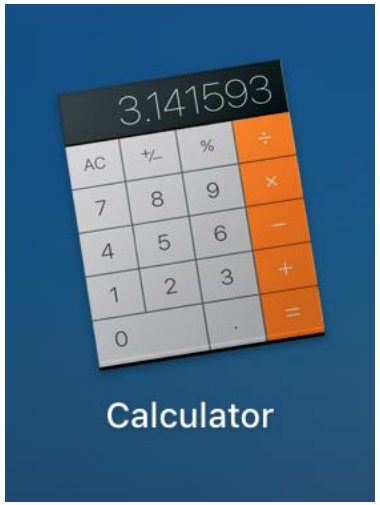
- A **reserved word** (or keyword) in Java is a **token** that has a predefined meaning in the language's grammar and cannot be redefined or used as an identifier.
- They are part of the Java specification.
- They are case-sensitive (e.g., Class is not the same as class).
- Using a reserved word as an identifier results in a compile-time error.

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Java's Primitive Data Types (whole numbers)

Type	Byte-size	Minimum value (inclusive)	Maximum value (inclusive)	Typical Use
byte	8-bit	-128	127	Useful in applications where memory savings apply.
short	16-bit	-32,768	32,767	
int	32-bit	-2,147,483,648	2,147,483,647	**Default** choice.
long	64-bit	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	Used when you need a data type with a range of values larger than that provided by int.



$2^8 = 256 = -128$ to $+127$

$2^{NumberOfBits}$	Range of values
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256

If the eight bit is used for the sign, we get a range from -128 to +127

i.e. 256/2 = +-128 values,
but a value is required to store 0, so range is **-128 to +127**

Declaring variables of a specific type

```
public class Week1Examples {
    public static void main(String[] args) {
        byte firstNumber; //declares a variable called firstNumber of type byte
        int secondNumber; //declares a variable called secondNumber of type int

        firstNumber = 40; //assign a value of 40 to firstNumber
        secondNumber = 78;

        System.out.println(firstNumber);  //print out the value

    }

}
```

Greyed out- indicates that the variable hasn't been used meaningfully

Declaring variables of a specific type

declaration

assignment

```
public class Week1Examples {
    public static void main(String[] args) {
        byte firstNumber; //declares a variable called firstNumber of type byte
        int secondNumber; //declares a variable called secondNumber of type int

        firstNumber = 40; //assign a value of 40 to firstNumber
        secondNumber = 78;

        int thirdNumber = 90;   //you can declare a variable and assign a
                                //value on one line.
```

Declaration and assignment in one

```
        System.out.println(firstNumber);  //print out the value

    }

}
```

Declaring variables of a specific type

```java
public class Week1Examples {
    public static void main(String[] args) {
        byte firstNumber; //declares a variable called firstNumber of type byte
        int secondNumber; //declares a variable called secondNumber of type int

        firstNumber = 40; //assign a value of 40 to firstNumber
        secondNumber = 78;

        int thirdNumber = 90;   //you can declare a variable and assign a
                                //value on one line.

        int x, y, z;          //multiple variables of the same type can be defined
                             //on one line.

        System.out.println(firstNumber);  //print out the value

    }

}
```

Declaring variables - some errors

```
int x, y, z;          //multiple variables of the same type can be defined
                     //on one line.
Int number1;
System.out.println(firstNumber);  //print out the value
```

RED - indicates a type of syntax error

Data types are case sensitive.

Int is not valid.
int is valid.

Declaring variables - some errors

```
int number = 60;
int number = 56;

System.out.println(firstNumber);  //print out the value
```

Syntax error – you cannot define two variables with the **same name**.

Declaring variables - some errors

Hover over the error and IntelliJ tries to help with the error.

```
int number = 60;
int number = 56;
```

Variable 'number' is already defined in the scope

Navigate to previous declared variable 'number' Alt+Shift+Enter More actions... Alt+Enter

```
int number = 56
```

Hello

```
System.o
```

Declaring variables - some errors

```
int x, y, z;          //multiple variables of the same type can be defined
                     //on one line.
int number = 60.57;

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

Syntax error:
you can only store whole numbers in an int variable.

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Java's Primitive Data Types (**decimal** numbers)

Type	Byte-size	Minimum value (inclusive)	Maximum value (inclusive)	Typical Use
float	32-bit	*Beyond the scope of this lecture .* *There is also a loss of precision in this data-type that we will cover in later lectures.*		Useful in applications where memory savings apply.
double	64-bit			Default choice when programming **Java apps.**

Declaring/defining a floating point field

- Use float or double for non-integer numbers.
- When assigning numbers, use the f or d suffix with float or double.

```
int x, y, z;          //multiple
                     //on one lir

float number = 60.57f;
double number2 = 60.57d;
```

Java's Primitive Data Types: **float** example

```java
public class Week1Example {
    public static void main(String[] args) {

        float xCoordinate = 14;
        float yCoordinate = 34;

    }
}
```

Whole numbers can be placed into a float variable.

Q: Why?

A: There is no loss of precision. We are not losing any data.

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Java's Primitive Data Types (others)

Type	Byte-size	Minimum value (inclusive)	Maximum value (inclusive)	Typical Use
char	16-bit	'\u0000' (or 0)	'\uffff' (or 65,535)	Represents a Unicode character.
boolean	1-bit	n/a		Holds either **true** or **false** and is typically used as a flag.

- We will go into more detail on these two data types in later lectures.

http://en.wikipedia.org/wiki/List_of_Unicode_characters

Java's Primitive Data Types (**memory sizes**) - Summary

	Data Type	Size (Bytes)	Size (Bits)
Whole numbers	byte	1	8
	short	2	16
	int	4	32
	long	8	64
Decimal numbers	float	4	32
	double	8	64
character	char	1	8
boolean	boolean		1

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Java's Primitive Data Types (**default values**)

	Data Type	Default Value
Whole numbers	byte	0
	short	0
	int	0
	long	0L
Decimal numbers	float	0.0f
	double	0.0d
character	char	'\u0000'
boolean	boolean	false

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Topics list

1. Variables.
2. Assignment statement.
3. Data Types.
4. Java's Primitive Data Types
    1. Whole numbers.
    2. Decimal numbers.
    3. Others.
5. Arithmetic operators.

Arithmetic Operators

Arithmetic Operator	Explanation	Example(s)
+	Addition	6 + 2 amountOwed + 10
-	Subtraction	6 – 2 amountOwed – 10
*	Multiplication	6 * 2 amountOwed * 10
/	Division	6 / 2 amountOwed / 10
%	Remainder	16 % 5 →1

Compound Assignment Statements

	Full statement	Shortcut
Mathematical shortcuts	x = x + a;	x += a;
	x = x - a;	x -= a;
	x = x * a;	x *= a;
	x = x/a;	x /=a;
Increment shortcut	x = x+1;	x++;
Decrement shortcut	x = x - 1;	x--;

Arithmetic Operators

- These examples are straightforward uses of the arithmetic operators.
- However, we typically want to do more complex calculations involving many arithmetic operators.
- To do this, we need to understand the **Order of Evaluation**.

Order of Evaluation

- **B**rackets **()**
- **M**ultiplication (*)
- **D**ivision (**/**)
- **A**ddition (**+**)
- **S**ubtraction (**-**)

BoMDAS

Beware My Dear Aunt Sally

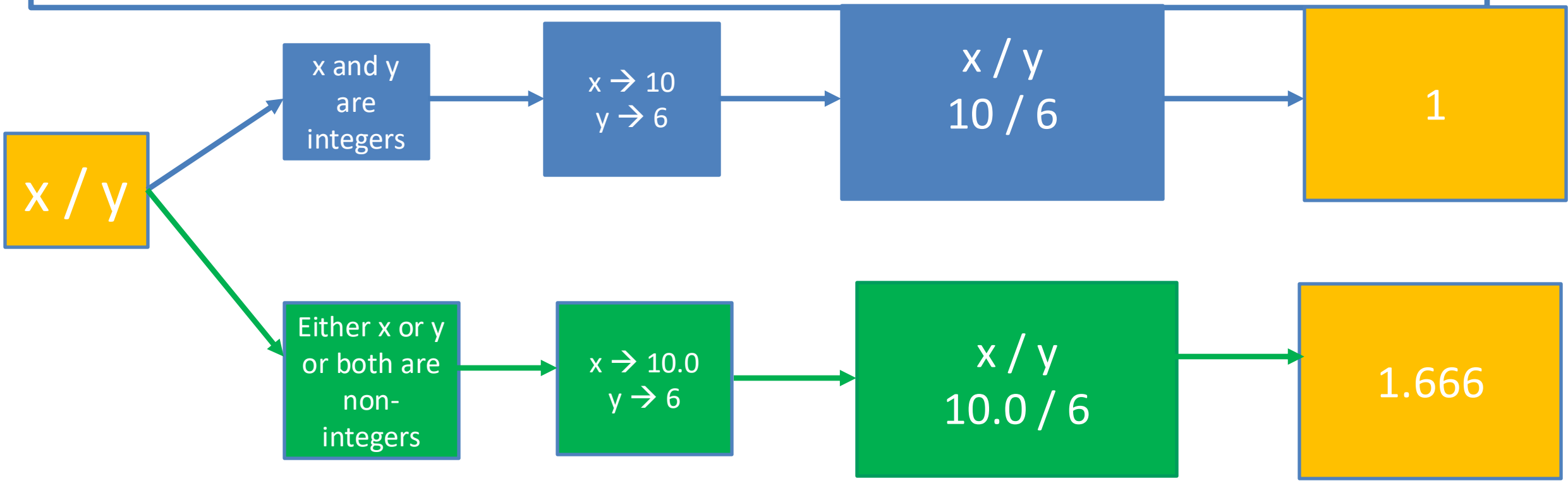
Order of Evaluation - **Quiz**

What are the results of these calculations?

- Q1: 3+6*5-2
- Q2: 3+6*(5-2)
- Q3: (3+6)*5-2

Note on division '/' – integer or 'normal' division

When we use the '/' operator , how it behaves depends on the type of the two operands :



Example 1

```
public class Week1Example {

    public static void main(String[] args) {

        int price = 500; //price stored in cents
        int quantity = 20;

        System.out.println("Price is " + price + "cents");
        System.out.println("Quantity is " + quantity);
        System.out.println("Total Price is " + price * quantity + "cents");

        System.out.println("Price is €" + (price/100));

    }
}
```

```
Price is 500cents
Quantity is 20
Total Price is 10000cents
Price is €5
```

Expected Console Output

Possible logic error in Example 1

- What happens if you change the price from 500 to 550?

```java
public class Week1Example {
    public static void main(String[] args) {

        int price = 550; //price stored in cents
        int quantity = 20;

        System.out.println("Price is " + price + "cents");
        System.out.println("Quantity is " + quantity);
        System.out.println("Total Price is " + price * quantity + "cents");

        System.out.println("Price is €" + (price/100));

    }
}
```

```
Price is 550cents
Quantity is 20
Total Price is 11000cents
Price is €5
```

Why does euro price not change?

Updated to include decimal division

```java
public class Week1Example {
    public static void main(String[] args) {

        int price = 550; //price stored in cents
        int quantity = 20;

        System.out.println("Price is " + price + "cents");
        System.out.println("Quantity is " + quantity);
        System.out.println("Total Price is " + price * quantity + "cents");

        System.out.println("Price is €" + (price/100.0));

    }
}
```

String concatenation

- 4 + 5

  9

- "wind" + "ow"

  "window"

- "Result: " + 6

  "Result: 6"

- "# " + price + " cents"

  "# 500 cents"

→ overloading

+ has two different functionalities

Quiz

- System.out.println(5 + 6 + "hello");

- System.out.println("hello" + 5 + 6);

Quiz

- System.out.println(5 + 6 + "hello");

  11hello

- System.out.println("hello" + 5 + 6);

  hello56

Formatted printing

- Concatenation can be used to create output in a desired format.
- An alternative is to use **`printf`**.

Questions?

Syntax Errors, Logic Errors & Comments in Java

Understanding mistakes and writing clearer code

Produced by:

Dr Siobhan Drohan
Mairead Meagher
Siobhan Roche
Mr Peter WIndle

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
setu.ie

Problem Solving

Programming IS problem solving.

Flow of Control in a Program

- Each program you write will typically have:

Sequence	Things that will be done in a particular order
Selection	Things that will be done conditionally
Iteration	Things that will be done repetitively

Flow of Control in a Program

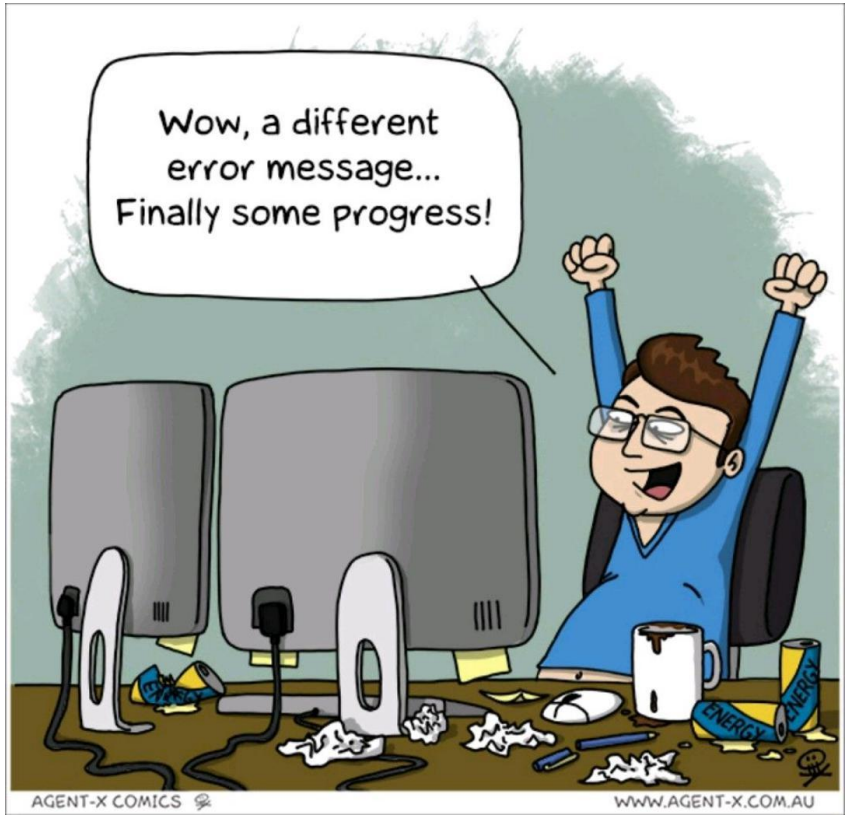
- Each program you write will typically have:

Sequence	Things that will be done in a particular order
Selection	Things that will be done conditionally
Iteration	Things that will be done repetitively

- Todays examples demonstrates ***Sequence***.
- We will cover ***Selection*** and ***Iteration*** in future weeks.

Why Errors Matter

- Programming is about **trial and error**
- Errors are not failures → they're feedback
- Three main categories:
  1. Syntax Errors – breaking the grammar rules → compiler won't run your code.
  2. Runtime Errors – code compiles but crashes while running.
  3. Logic Errors – code runs but produces the wrong result.



Syntax and Syntax Errors

- You will have seen the term Syntax mentioned before.
- Syntax are the rules you must follow when writing well-formed statements in a programming language.
- When you don’t follow the rules, Java will not run your code; instead you will get an error.

Syntax Errors – What are they?

- Definition:

  Errors in **code structure/grammar** that prevent the program from compiling
- Examples in Java:
  - Missing semicolon ;
  - Using Int instead of int
  - Unmatched braces { }
- Compiler usually highlights them immediately

Syntax Errors- Example

```
public class Example {
    public static void main(String[] args) {
        int number = 10
        System.out.println(number);
    }
}
```

Missing ; after 10 → compiler error

Logic Errors – What Are They?

In computer programming, a **logic error** is a bug in a program that causes it to operate incorrectly, but not to terminate abnormally (or crash). A **logic error** produces unintended or undesired output or other behaviour, although it may not immediately be recognised as such.

Logic error - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Logic_error**

The program runs, but gives the wrong result

- Harder to spot than syntax errors
- Examples:
  - Using integer division instead of floating point
  - Incorrect formula for calculation
  - Wrong variable used in an expression

Logic Error Example

```
public class Example {
    public static void main(String[] args) {
        int price = 550; // cents
        System.out.println("Price in euro: " + price / 100);
    }
}
```

Output: Price in euro: 5 ❌
Correct should be 5.50 → fix with price / 100.0

How to spot Logic Errors

- Careful testing with different values
- Using print statements to trace variables
- Later: debugging tools in IntelliJ

Runtime Errors

- Definition:

  Errors that occur while the program is running, after successful compilation.
- Cause the program to stop abruptly (crash) with an exception.
- Common causes:
  - Division by zero
  - Accessing array elements out of bounds
  - Null references (using an object that hasn’t been created)
  - Invalid user input

Runtime Error - Example

```
public class RuntimeExample {
   public static void main(String[] args) {
      int x = 10;
      int y = 0;
      System.out.println(x / y); //  ArithmeticException can’t divide by 0
   }
}
```

Introduction to Comments

- Comments are notes for humans, ignored by the compiler

- Types:
  - Single-line comment //
  - Multi-line comment /* ... */
  - Javadoc comment /** ... */

Commenting your code…

```
//  This is a comment.
// Anything typed after the two slashes
// up to the end of the line, is ignored by Java.

/*  This is a longer comment.  As you can span more than one line
with this comment style, it can be quite handy. */
```

Why use Comments?

- Explain the purpose of code
- Help future you (and teammates) understand logic
- Used for documentation in professional projects

Code Example

```
public class SalaryCalculator {
  public static void main(String[] args) {
    // Hourly rate in euros
    double hourlyRate = 12.5;

    /* Hours worked in a week
      (including overtime) */
    int hoursWorked = 45;

    // Calculate weekly pay
    double totalPay = hourlyRate * hoursWorked;

    System.out.println("Weekly Pay: €" + totalPay);
  }
}
```

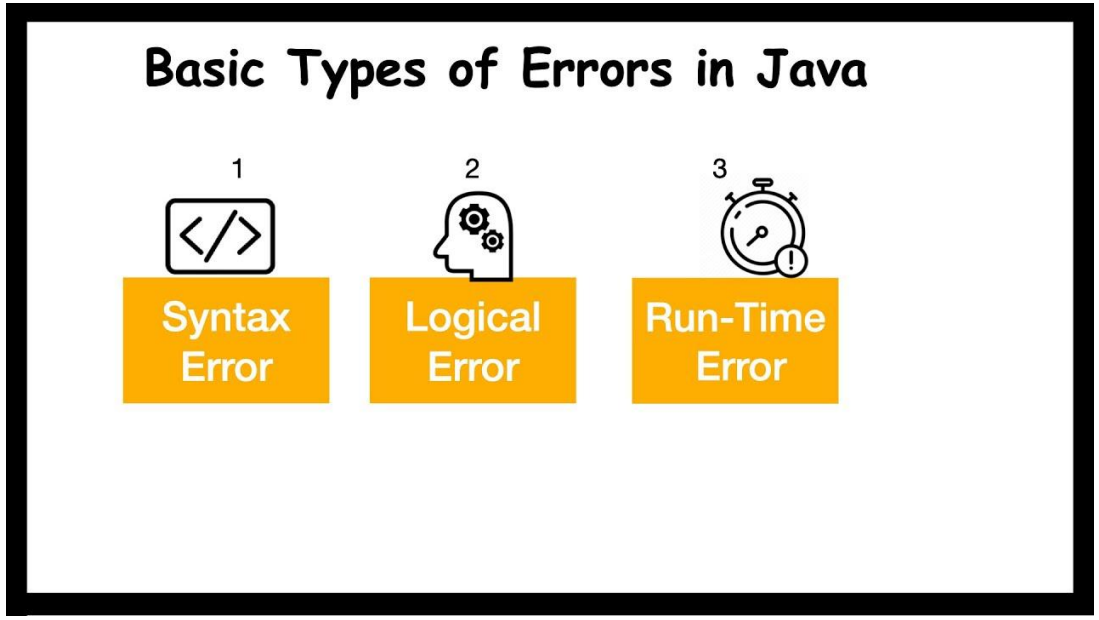
- // single line comment
- /* */ multiline comment

Best practices for comments

- Don’t state the obvious (int x = 5; // set x to 5)
- Write why not just what
- Keep comments up to date with code

Summary

Syntax errors: compiler catches them
Logic errors: harder, need testing/debugging
Runtime errors: can crash your program
Comments: make code readable & maintainable



Questions?