Classes

Writing your own classes

How have we written code so far?

- Everything in the ”main” method.
- Following instructions from top to bottom inside the main method.

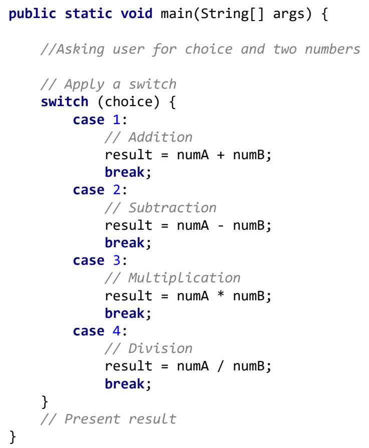
For how long is this feasible?

As the complexity of our applications grow, this style will quickly lead to unmaintainable and messy code.

Writing your own classes

For example if we write a simple calculator

We might end up with a switch resembling:



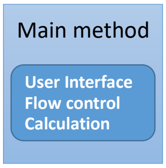
```
public static void main(String[] args) {

    //Asking user for choice and two numbers

    // Apply a switch
    switch (choice) {
        case 1:
            // Addition
            result = numA + numB;
            break;
        case 2:
            // Subtraction
            result = numA - numB;
            break;
        case 3:
            // Multiplication
            result = numA * numB;
            break;
        case 4:
            // Division
            result = numA / numB;
            break;
    }
    // Present result
}
```

Writing your own classes

The Problem

The main method has all the responsibility for:

- **User interaction**
  (printing and reading to and from the console)
- **Mathematics**
  (Calculating the expressions)

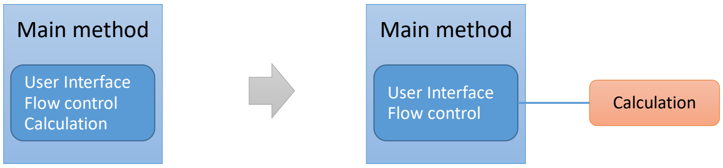


If either of these change, we have to change the method.
We cannot re-use code elsewhere.

Writing your own classes

An alternative: Separating Responsibility

Let’s start by breaking off the calculation logic into a separate class.



This means we will have **fewer reasons** to change a given block of code, it will be easier to maintain, and we can use the same module another place if we need it.

This principle is known as **separation of concerns**

Writing your own classes

We have used many of the built-in **types** through the JDK base class library, including:

```java
int x = 42;

String company = "Edument AB";

StringBuilder sb = new StringBuilder("Hello ");

LocalDateTime date1 = LocalDateTime.of(2016, 9, 19, 14, 45, 00);
```

We can also define our own types. One way of doing this is by using the **class** keyword

Writing a class

A class can be seen as the blueprint of your own data type.

An empty class:

```
public class Calculator {

}
```

Tip: Class names should always begin with a capital letter.

Writing a class

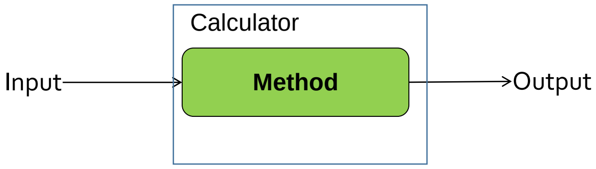
We can declare a variable of this type:

```
public class Calculator {

}

public class Main {
    public static void main(String[] args) {
        // This compiles, but we can’t really
        // do anything with it:
        Calculator calculator;
    }
}
```

Note the difference in the casing.

Adding behaviour - Methods

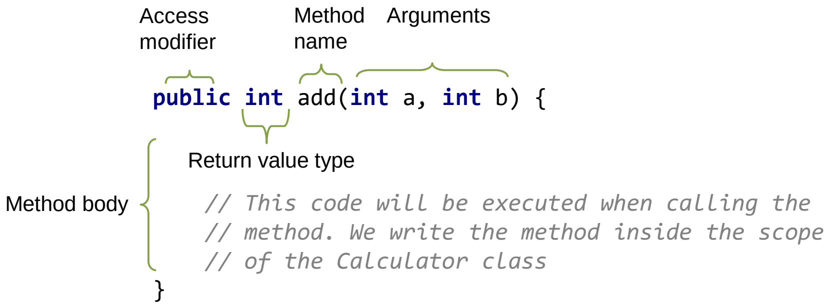
In order to do something useful with the class, we have to give it some kind of **behaviour**. One way of doing this is through **methods**:



We give input to the **method**, which in turn produces output. The method is a part of the **Calculator class**

Method syntax

Example: adding two numbers.



```
public int add(int a, int b) {

    // This code will be executed when calling the
    // method. We write the method inside the scope
    // of the Calculator class
}
```

The **arguments** are the input to the method, the **return value** type specifies what we’re expecting back.

Method syntax

Example: adding two numbers.

```
public int add(int a, int b) {
    // Calculate the sum and return it
    return a + b;
}
```

Any method with an expected return value **must** have a return statement for every path through the method.

Tip: Method names should always begin with a lower case letter.

Method syntax

Some methods don’t need to return anything.

These are marked with the keyword **void** return type.

```
public void DoSomething(String input) {
    // No return statement needed
}
```

Class syntax

Our calculator class so far:

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
```

A class can contain as many methods as we like.

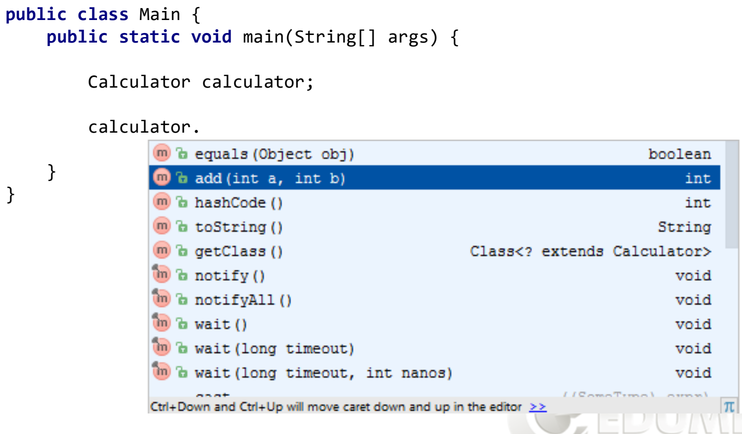
The entire class

Implementing the rest of the methods would result in:

```
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }

    public int subtract(int a, int b) {
        return a-b;
    }

    public int multiply(int a, int b) {
        return a*b;
    }

    public float divide(int a, int b) {
        return a/b;
    }
}
```

Class syntax

We already get IntelliSense for the method we have added to the Calculator class.



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

        Calculator calculator;

        calculator.
    }
}
```

```
equals(Object obj)                                boolean
add(int a, int b)                                     int
hashCode()                                            int
toString()                                         String
getClass()                   Class<? extends Calculator>
notify()                                             void
notifyAll()                                          void
wait()                                               void
wait(long timeout)                                   void
wait(long timeout, int nanos)                        void
cast                                     ((SomeType) expr)
```

Ctrl+Down and Ctrl+Up will move caret down and up in the editor >>

Using your classes

Declaration

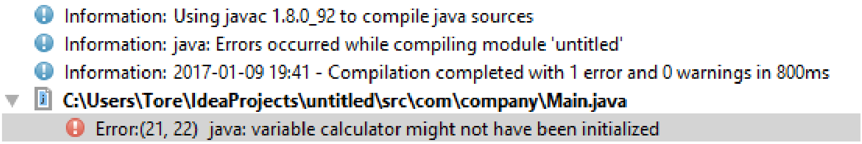
We already get IntelliSense for the method we have added to the Calculator class.

```
public static void main(String[] args) {
    Calculator calculator;
    int result = calculator.add(5, 6);
}
```

- According to IntelliSense, there is a method called ”add”.
- In this case, we expect **result** to be 5
- So why won’t this code compile?

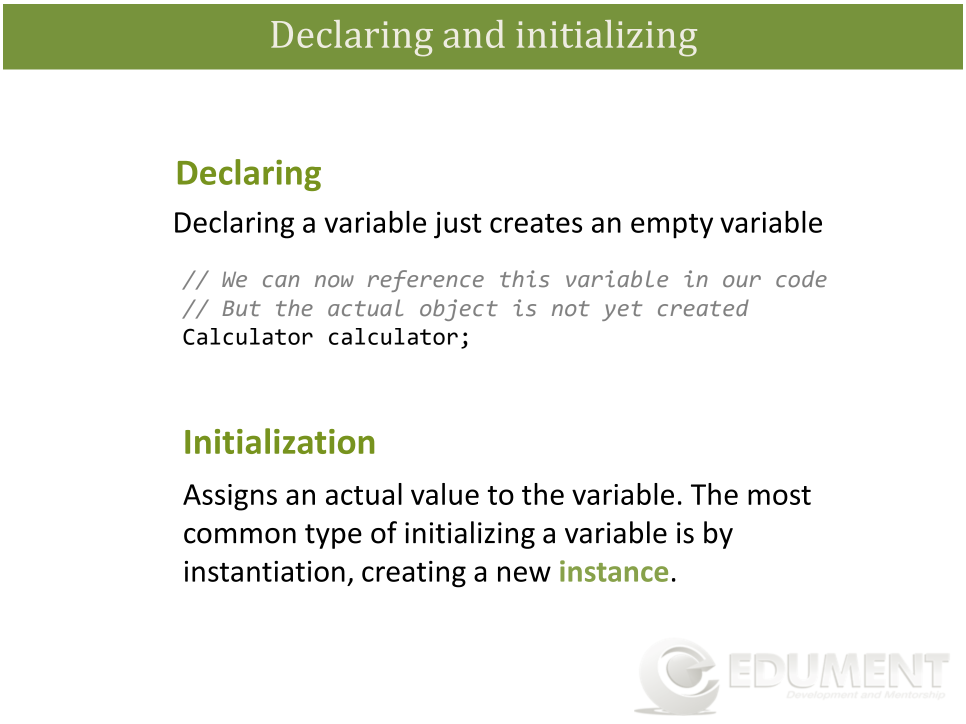
Declaration

Declaring isn’t enough!



```
Information: Using javac 1.8.0_92 to compile java sources
Information: java: Errors occurred while compiling module 'untitled'
Information: 2017-01-09 19:41 - Compilation completed with 1 error and 0 warnings in 800ms
C:\Users\Tore\IdeaProjects\untitled\src\com\company\Main.java
    Error:(21, 22)  java: variable calculator might not have been initialized
```

From one of today’s first slides:
To just **declare** a variable is generally not enough.
In order to use it, we have to **assign** a value to it as well.



Declaring and initializing

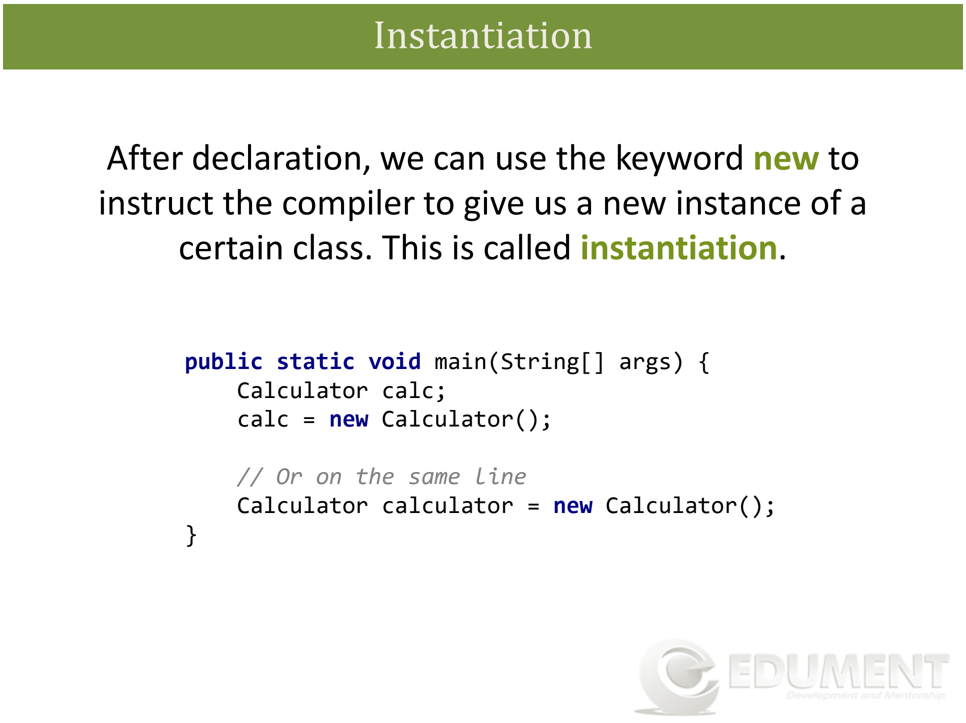
Declaring

Declaring a variable just creates an empty variable

```
// We can now reference this variable in our code
// But the actual object is not yet created
Calculator calculator;
```

Initialization

Assigns an actual value to the variable. The most common type of initializing a variable is by instantiation, creating a new **instance**.



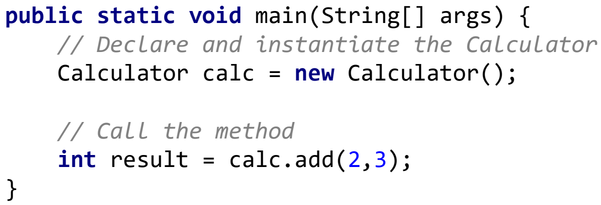
Instantiation

After declaration, we can use the keyword **new** to instruct the compiler to give us a new instance of a certain class. This is called **instantiation**.

```
public static void main(String[] args) {
    Calculator calc;
    calc = new Calculator();

    // Or on the same line
    Calculator calculator = new Calculator();
}
```

Revised code

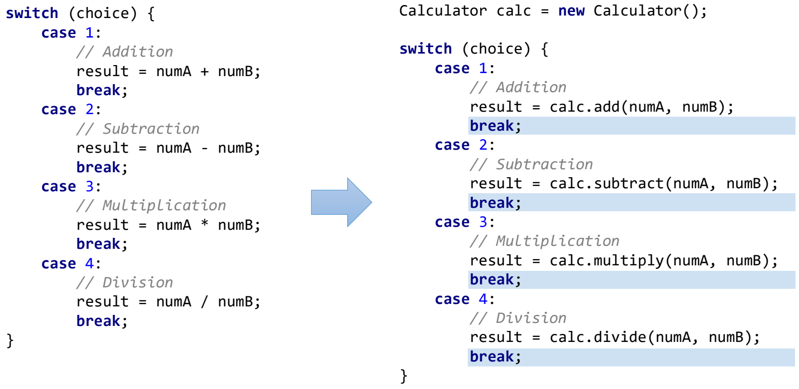
A revised version of our code:



```
public static void main(String[] args) {
    // Declare and instantiate the Calculator
    Calculator calc = new Calculator();

    // Call the method
    int result = calc.add(2,3);
}
```

Revisiting Exercise 5

The **switch** in our earlier example can now be written.



```
switch (choice) {
    case 1:
        // Addition
        result = numA + numB;
        break;
    case 2:
        // Subtraction
        result = numA - numB;
        break;
    case 3:
        // Multiplication
        result = numA * numB;
        break;
    case 4:
        // Division
        result = numA / numB;
        break;
}
```

```
Calculator calc = new Calculator();

switch (choice) {
    case 1:
        // Addition
        result = calc.add(numA, numB);
        break;
    case 2:
        // Subtraction
        result = calc.subtract(numA, numB);
        break;
    case 3:
        // Multiplication
        result = calc.multiply(numA, numB);
        break;
    case 4:
        // Division
        result = calc.divide(numA, numB);
        break;
}
```

The main method no longer has responsibility for doing the calculations: we have separated our **business logic** from our **presentational logic**

Fields

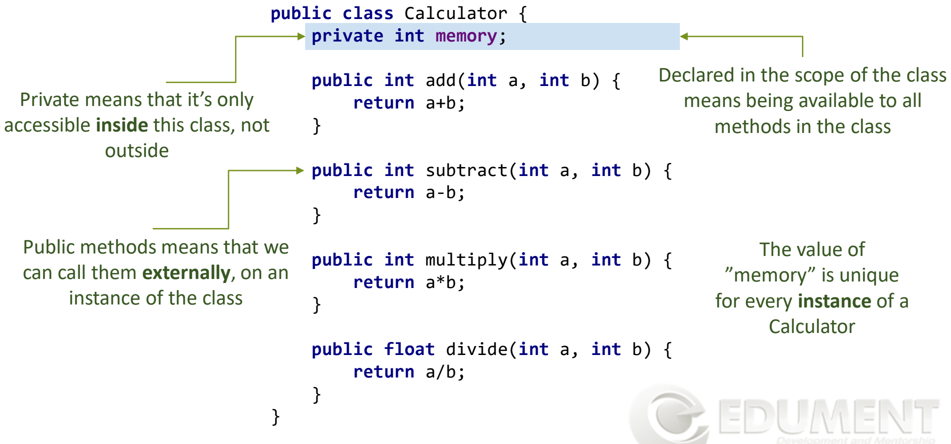
Fields (1)

As previously discussed, a variable is only accessible in the scope where it is defined. We have, so far, only defined variables in the **main** method.

```
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }

    public int subtract(int a, int b) {
        return a-b;
    }

    public int multiply(int a, int b) {
        return a*b;
    }

    public float divide(int a, int b) {
        return a/b;
    }
}
```

Fields (2)

Variables can be declared within the scope of a **class** as well. This is called a **field** or a **member variable**:



```
public class Calculator {
    private int memory;

    public int add(int a, int b) {
        return a+b;
    }

    public int subtract(int a, int b) {
        return a-b;
    }

    public int multiply(int a, int b) {
        return a*b;
    }

    public float divide(int a, int b) {
        return a/b;
    }
}
```

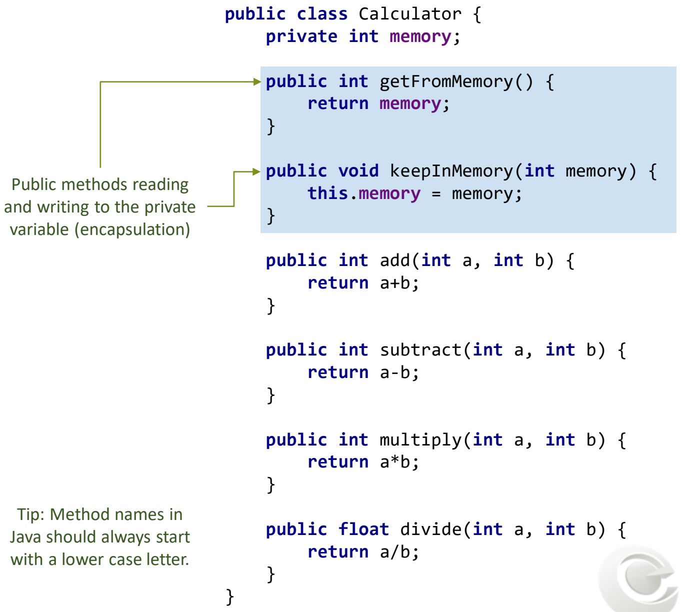
Private means that it’s only accessible **inside** this class, not outside

Declared in the scope of the class means being available to all methods in the class

Public methods means that we can call them **externally**, on an instance of the class

The value of ”memory” is unique for every **instance** of a Calculator

Fields (3)



```
public class Calculator {
    private int memory;

    public int getFromMemory() {
        return memory;
    }

    public void keepInMemory(int memory) {
        this.memory = memory;
    }

    public int add(int a, int b) {
        return a+b;
    }

    public int subtract(int a, int b) {
        return a-b;
    }

    public int multiply(int a, int b) {
        return a*b;
    }

    public float divide(int a, int b) {
        return a/b;
    }
}
```

Public methods reading and writing to the private variable (encapsulation)

Tip: Method names in Java should always start with a lower case letter.

Exercise 10

Let's do exercise 10