Methods in Classes and Objects

PREVIEW / RECAP:

We have seen various variable types which can be used in classes and objects.
When and how to use static and non - static variable

What is Method?

A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything
Methods are **time savers** and help us to **reuse** the code without retyping the code.

Method Declaration

In general, method declarations has six components :

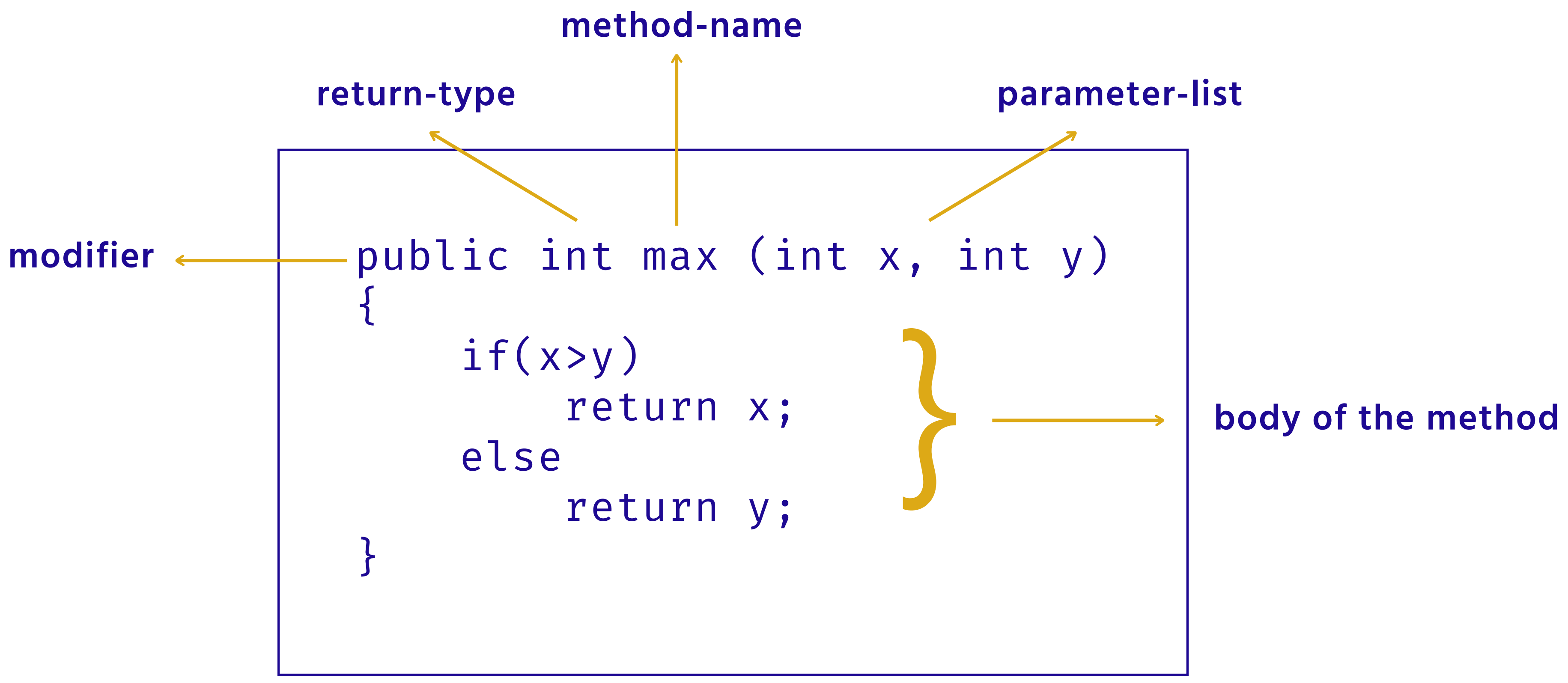
1- Modifier : Defines access type of the method i.e. from where it can be accessed in your application. In Java, there are 4 types of the access specifiers.

- public: accessible in all classes in your application.
- protected: accessible within the class in which it is defined and in its subclass(es)
- private: accessible only within the class in which it is defined.
- default (declared/defined without using any modifier) : accessible within the same class and package within which its class is defined.

MODIFIER CONCEPT IS EXPLAINED BRIEFLY UNDER ENCAPSULATION TOPIC

2- The return type : The data type of the value returned by the method or void if does not return a value.
3- Method Name : Name of the method, using which we can call a particular method. The rules for field names apply to method names as well, but the convention is a little different.

4- Parameter list : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parentheses. If there are no parameters, you must use empty parentheses ().
5- Exception list : The exceptions you expect by the method can throw, you can specify these exception(s).
6- Method body : it is enclosed between braces. The code you need to be executed to perform your intended operations.



```
public int max (int x, int y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

Method signature

It consists of the method name and a parameter list (number of parameters, type of the parameters and order of the parameters). The return type and exceptions are not considered as part of it.

Method Signature of above function:

max(int x, int y)

Creating Method

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
    // method body
}
```

The syntax shown above includes −

- **modifier −** It defines the access type of the method and it is optional to use.
  (IF U WANT TO LEARN ABOUT MODIFIERS IN DETAIL IT IS COVERED IN ENCAPSULATION TOPIC)
- **returnType −** Method may return a value.
- **nameOfMethod −** This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List −** The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body −** The method body defines what the method does with the statements.

Considering the following example to explain the syntax of a method −

```
public static int methodName(int a, int b) {
  // body
}
```

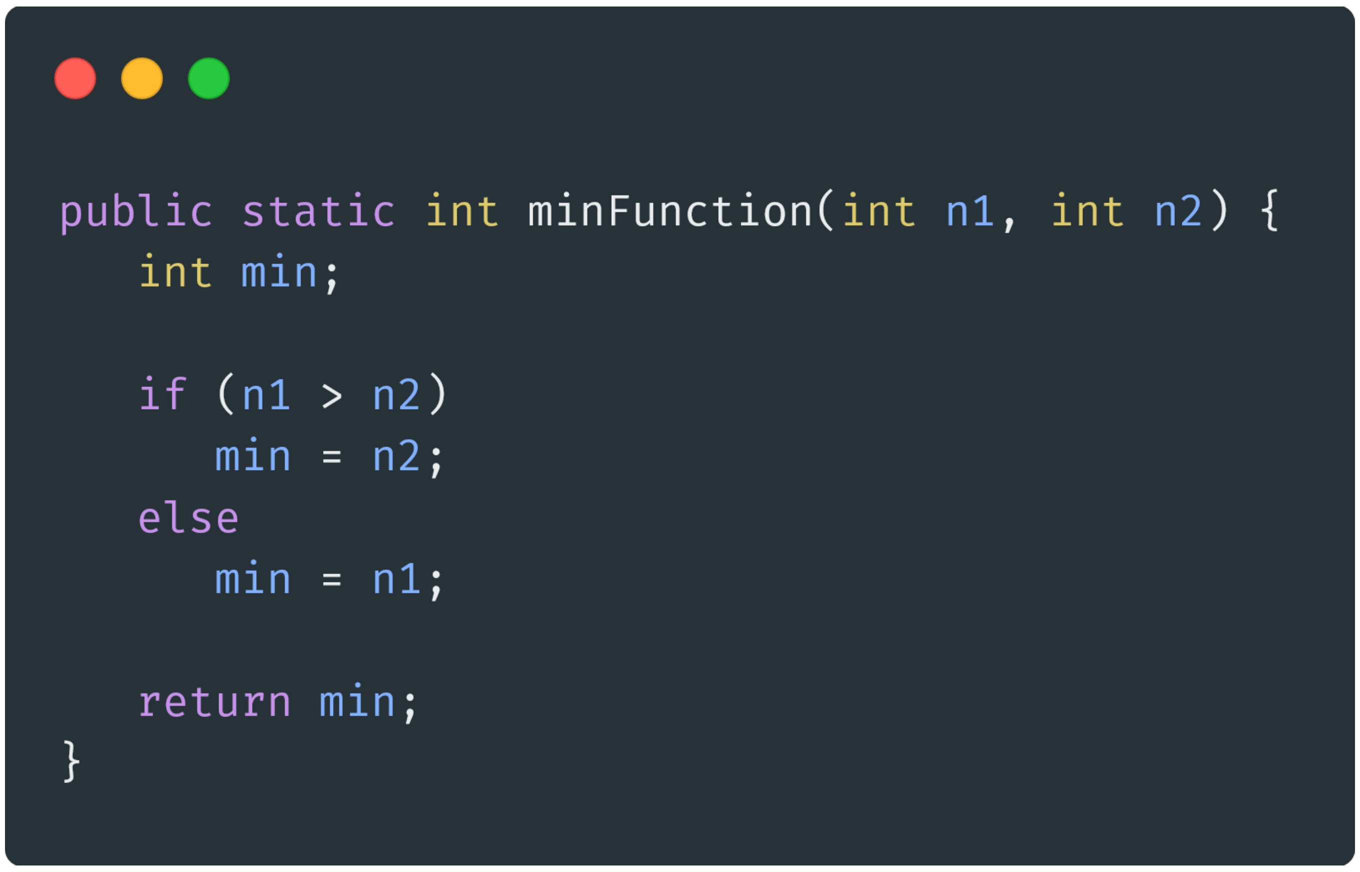
Here,

- **public static −** modifier
- **int −** return type
- **methodName −** name of the method
- **a, b −** formal parameters
- **int a, int b −** list of parameters

Now u must have understood it better. Let's take an example of resting a method to find a minimum of 2 numbers.

E.g

Here is the source code of the method called min(). This method takes two parameters num1 and num2 and returns the minimum between the two –
/** the snippet returns the minimum between two numbers */



```java
public static int minFunction(int n1, int n2) {
   int min;

   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Great! Now we have understood how to create a method. However we will also have to take a look at how to call methods.

Calling a Method

For using a method, it should be called. The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method.

There are two ways in which a method is called

1. a method returns a value

We have to specify the return type of a method at the time method creation, if it is returning some value.
This called method then returns control to the caller, when the return statement is executed.

e.g.

```
public static int methodName( int b) {
   return b;
   System.out.println(b);
}
```

NOTE: - Above method will not print value of b when called. As it shifts control to the caller after "return" statement.

2. returns nothing (no return value).

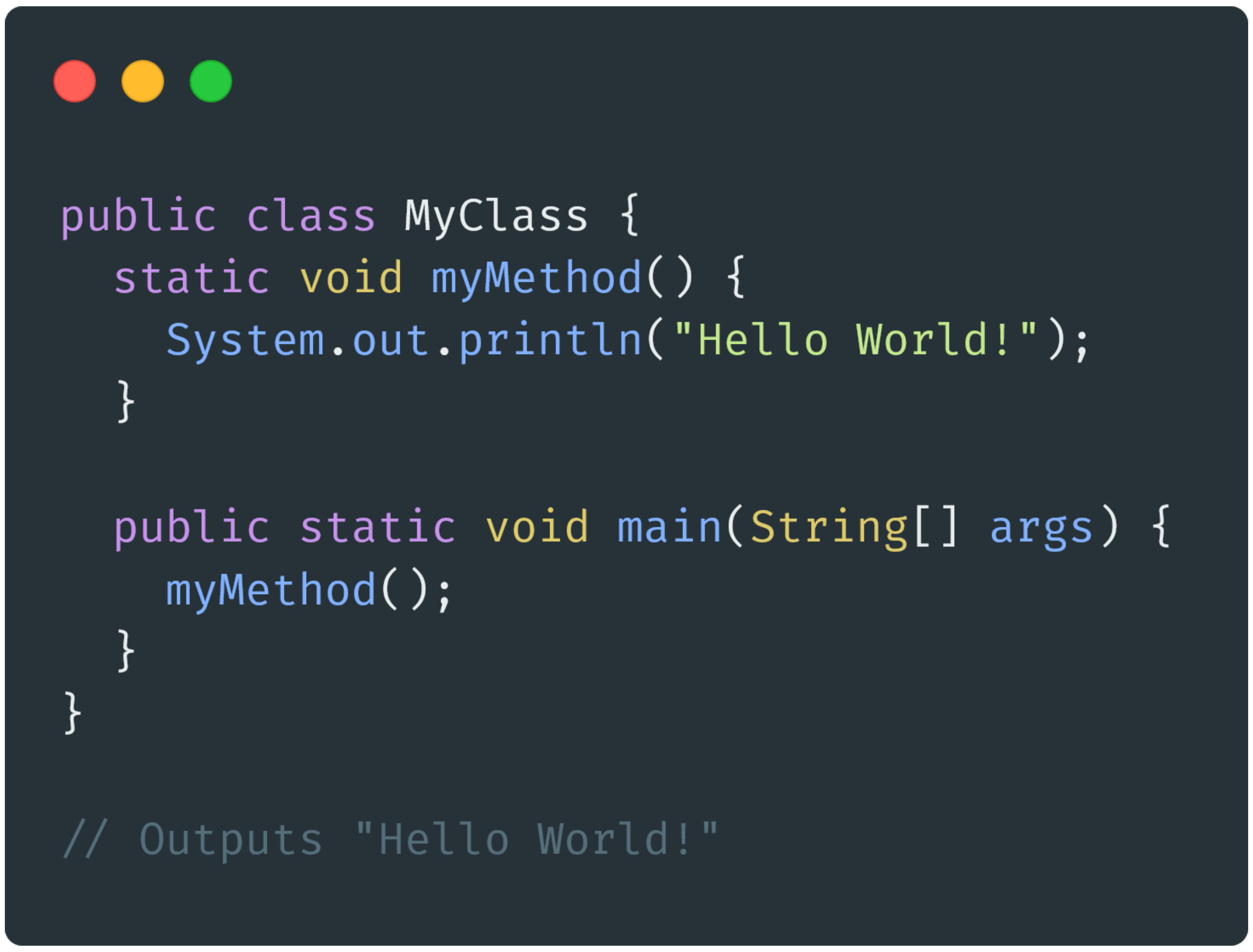
We simply have to write **void** as a return type of method, if it is not going to return any value.
This called method then returns control to the caller, when it reaches the method ending closing brace.

```
public static void methodName( int b) {
   System.out.println(b);
}
```

To call a method, write the method's name followed by two parentheses () and a semicolon;

e.g.

Inside main, call myMethod():



```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}

// Outputs "Hello World!"
```

Ohh Great! Now we also know how to call a user-defined method.

Do You Know?

1. **User-defined method:** Any method which is created by user, is called as user-defined method
2. **Inbuilt method:** Any method which is not created by user, is called as inbuilt method
   E.g. System.out.println() is inbuilt method

Ok!!! So Now you have some knowledge about methods. But have you remember in variables section we have learned about static and non-static variables.
So methods can also be declared as static or non-static. Let's see how we can do this ;)

Static Methods

This method's definition starts with a “static” keyword. Static methods **CAN BE** called without creating object of class.

```java
public class MyClass {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    MyClass myObj = new MyClass(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

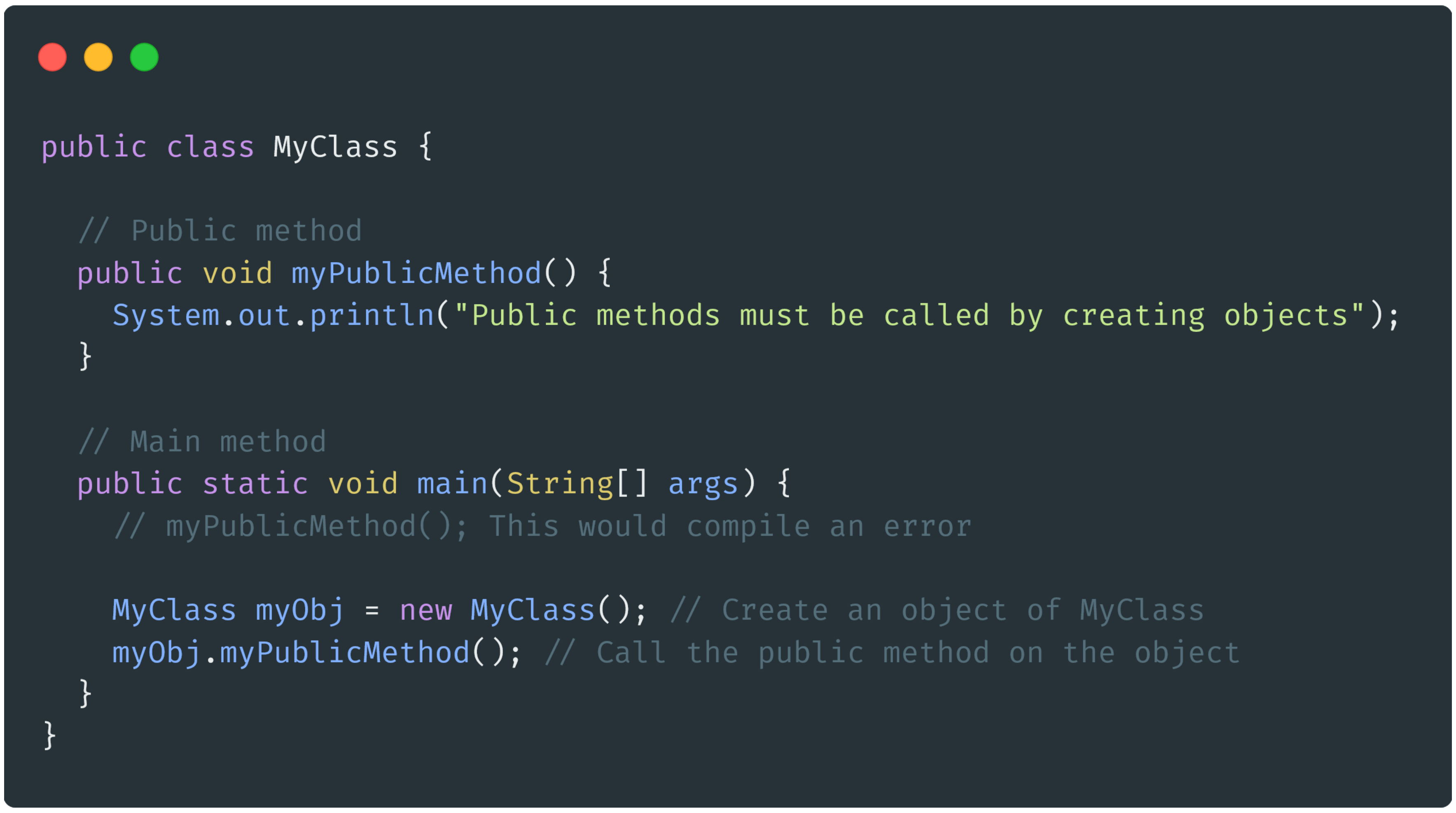
Output :

```
Static methods can be called without creating objects
Static methods can be called without creating objects
```

Non-static methods:

This method starts with a keyword public. Non-Static methods **MAY BE** called without creating an object of class.

e.g.

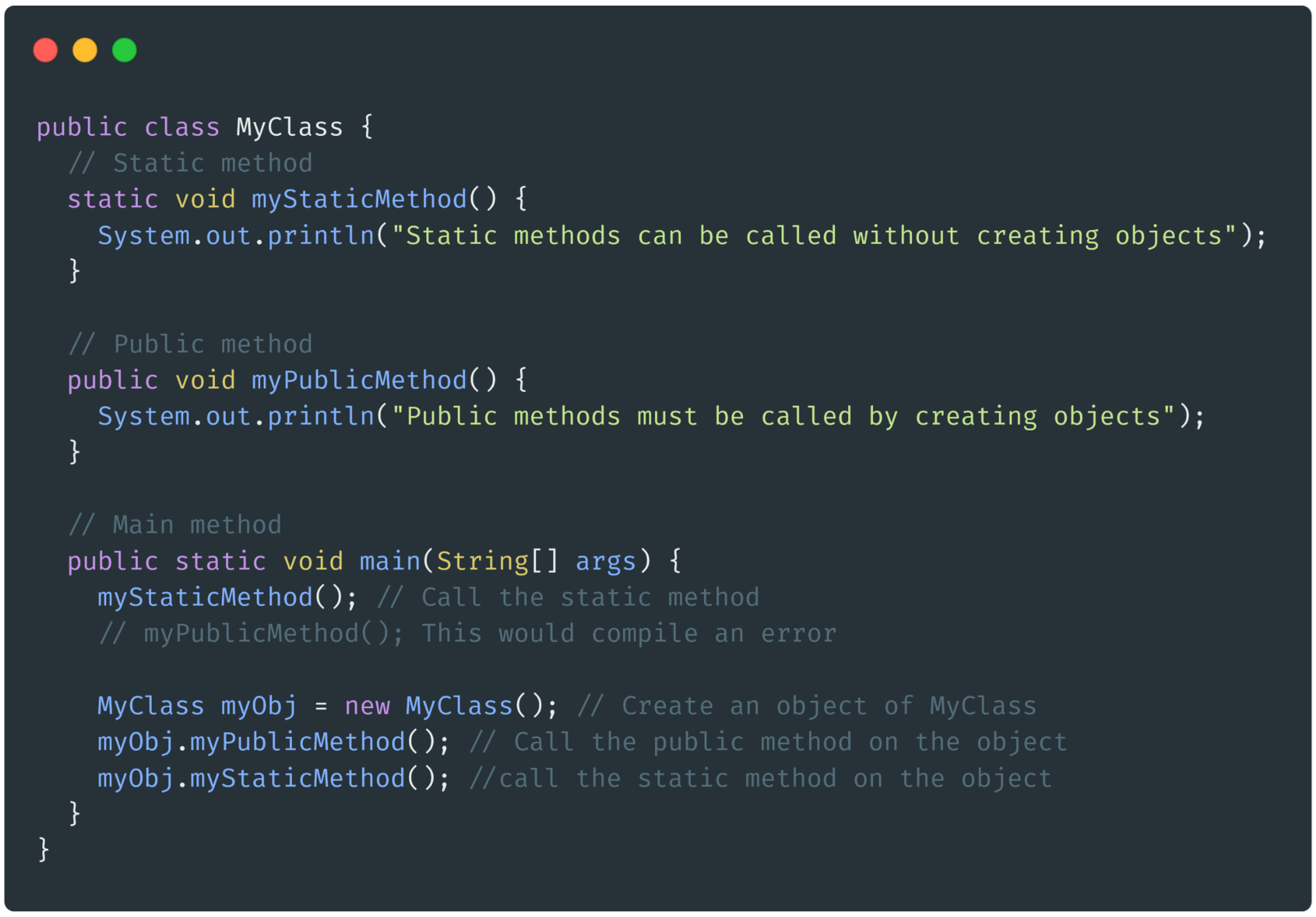


```java
public class MyClass {

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    // myPublicMethod(); This would compile an error

    MyClass myObj = new MyClass(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Output: -

Public methods must be called by creating objects

Ummm... still have a confusion in mind about what is difference in static and non-static methods. Let's combine above 2 codes of static method example and non-static example...



```java
public class MyClass {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    MyClass myObj = new MyClass(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method on the object
    myObj.myStaticMethod(); //call the static method on the object
  }
}
```

Output

```
Static methods can be called without creating objects
Public methods must be called by creating objects
Static methods can be called without creating objects
```

Great!! Now you have the knowledge of Method declaration, Method calling, static/non-static methods.
But there is something more to know about methods: - Method Overriding and Method Oveloading. We learn about this next part