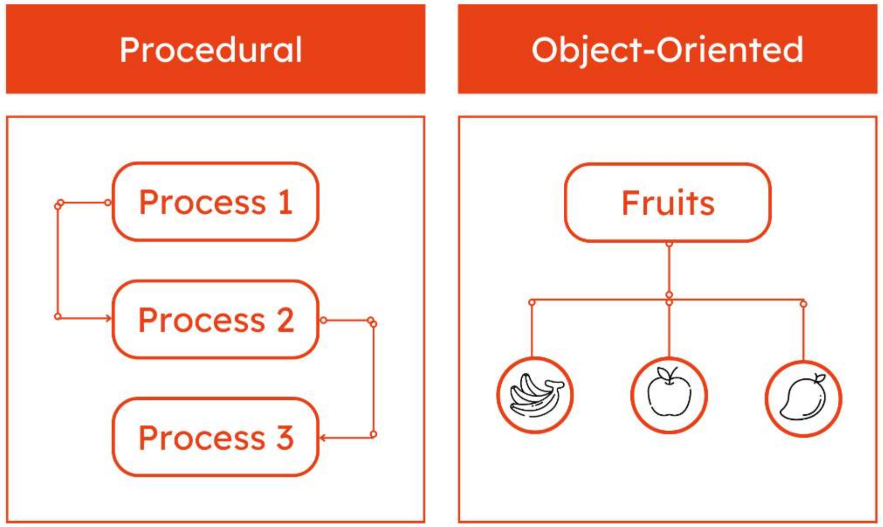
Java Programming
Chapter 7

Object Oriented Programming

OOP

- OOP stands for **Object-Oriented Programming**.
- Programming Tech.
  - Procedural programming
    - C / pascal / fotran
  - OOP
    - Java / c++



OOP

- Advantages of OOP
  - OOP is faster and easier to execute
  - OOP provides a clear structure for the programs
  - OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
  - OOP makes it possible to create full reusable applications with less code and shorter development time

"Don't Repeat Yourself"

➔ reducing the repetition of code

Key Concepts

1. Encapsulation
    - The bundling of data (properties) and the methods that operate on that data into a single unit, or object.
    - This helps to hide the internal state and complexity from the outside world.

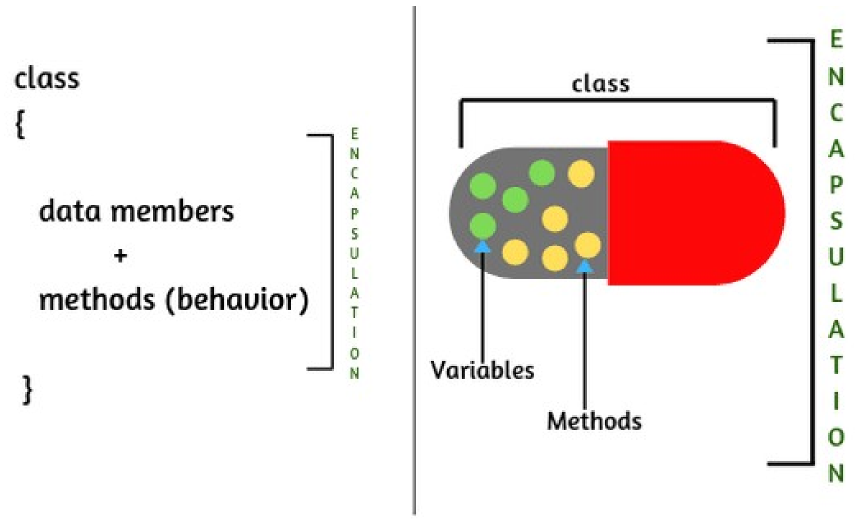
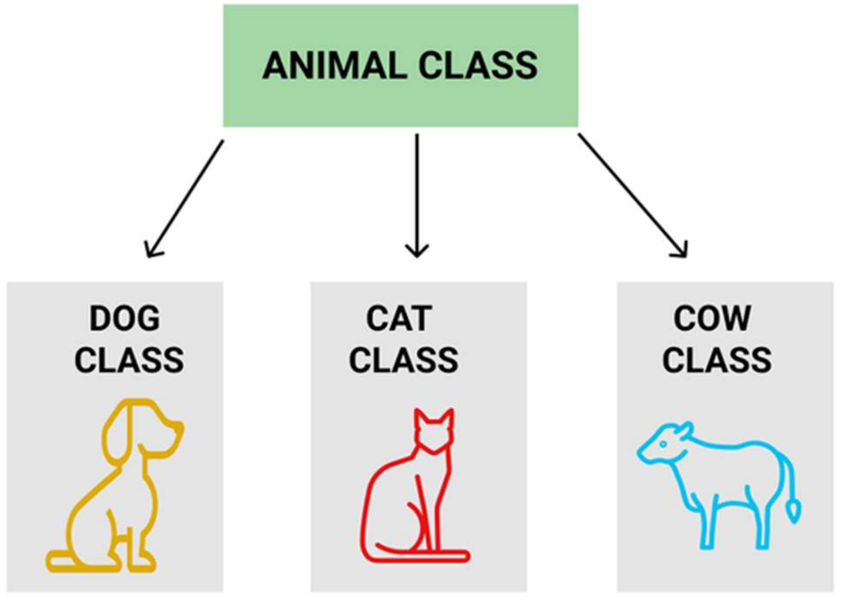


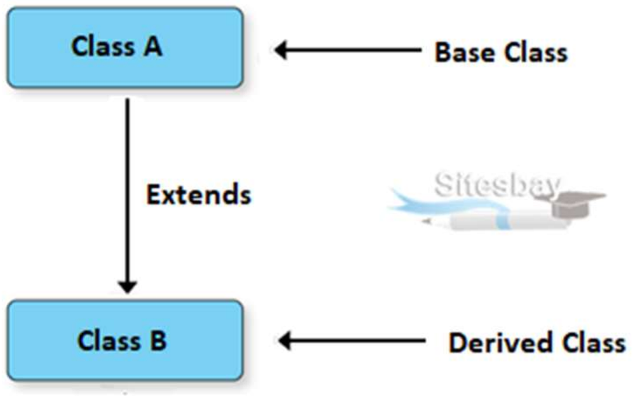
Fig: Encapsulation

Key Concepts

2. Inheritance

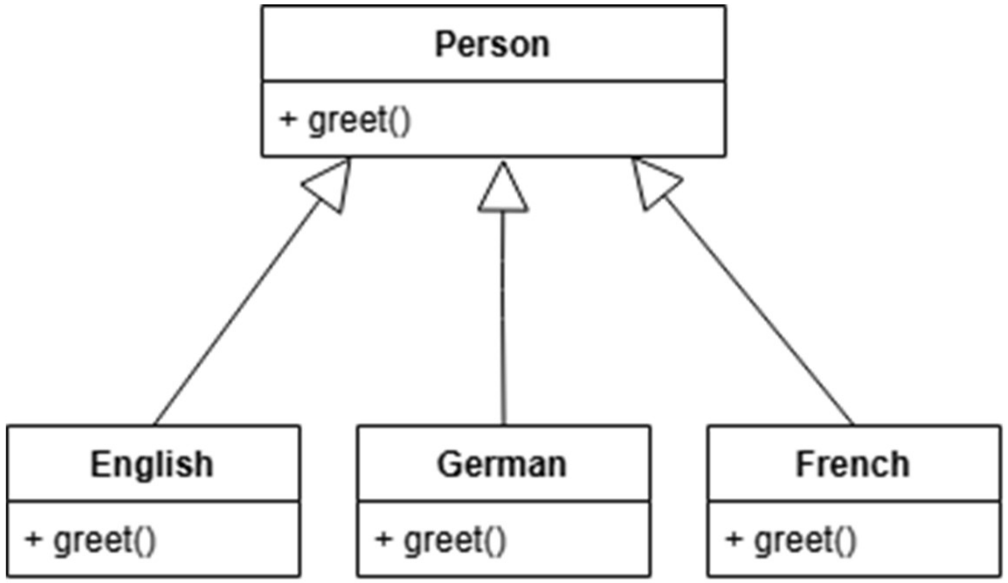
- A mechanism where a new class can inherit the properties and methods of an existing class. This promotes code reusability.





Key Concepts

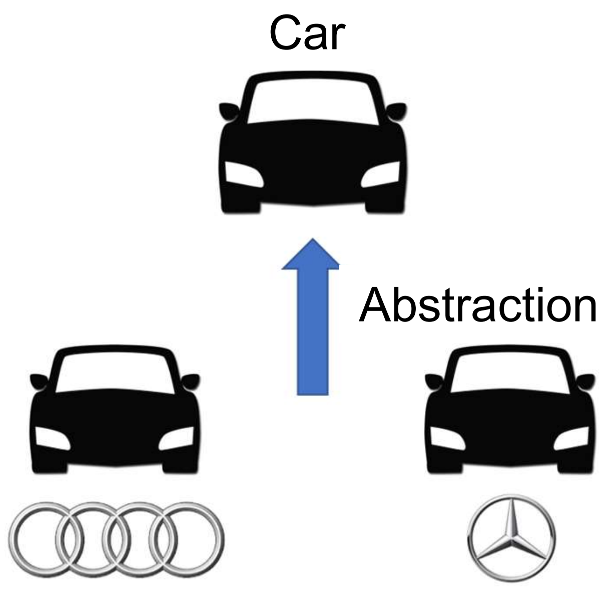
3. Polymorphism
   - The ability of an object to take on many forms.
   - It allows you to treat objects of different classes in a uniform way, as long as they share a common interface or superclass.

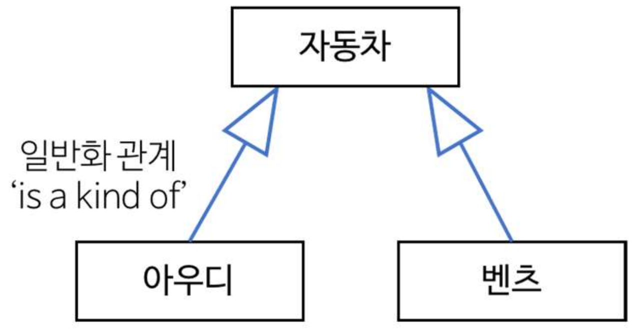


Key Concepts

4. Abstraction

- The process of representing essential features without including background details.
- It focuses on what an object does rather than how it does it.





Class vs Object

- Classes and objects are the two main aspects of object-oriented programming.

Class

Fruit

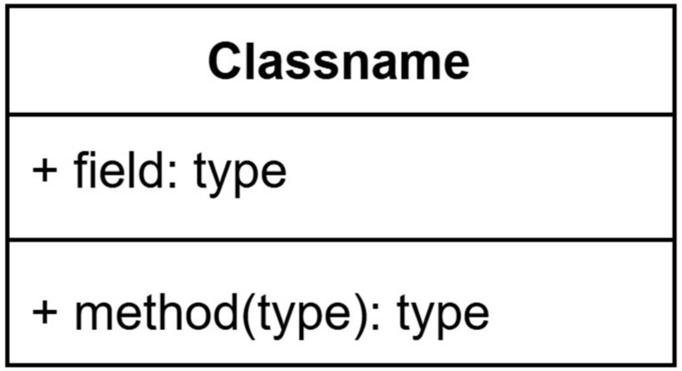
Objects

Apple
Banana
Mango

- class is a template for objects, and an object is an instance of a class.

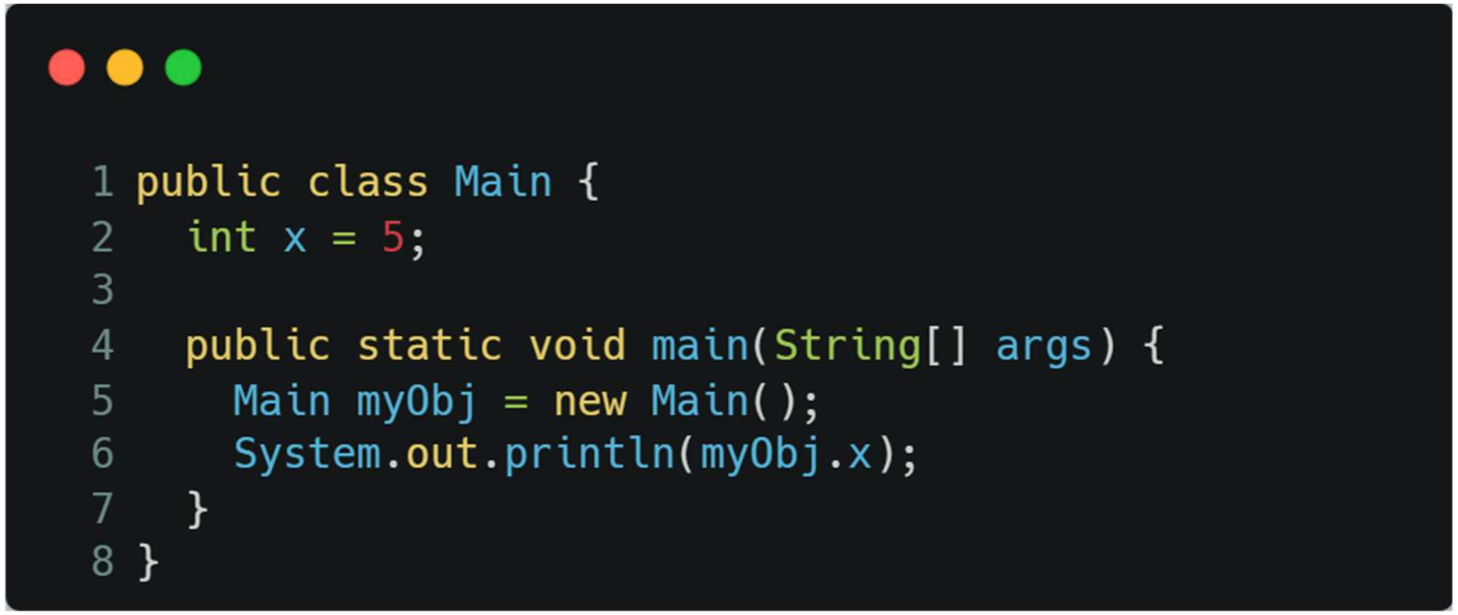
Class vs Object

- Everything in Java is associated with classes and objects, along with its attributes and methods
- For example: in real life,
  - a car is an object.
  - The car has **attributes**, such as weight and color,
  - **Methods**, such as drive and brake.
- Class Diagram



Class

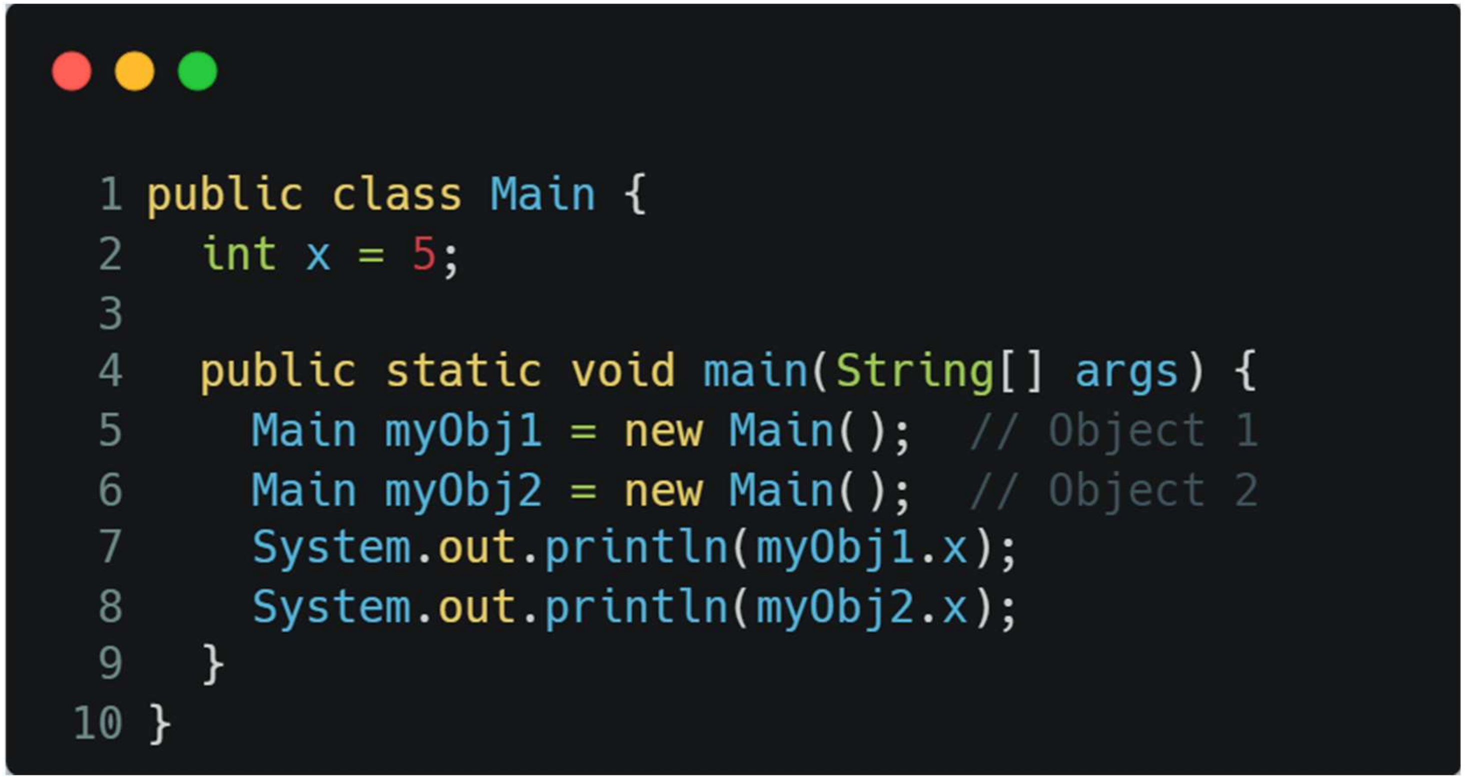
- In Java, an object is created from a class
- To create an object of Main, specify the class name, followed by the object name, and use the keyword ***new***



```java
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Class

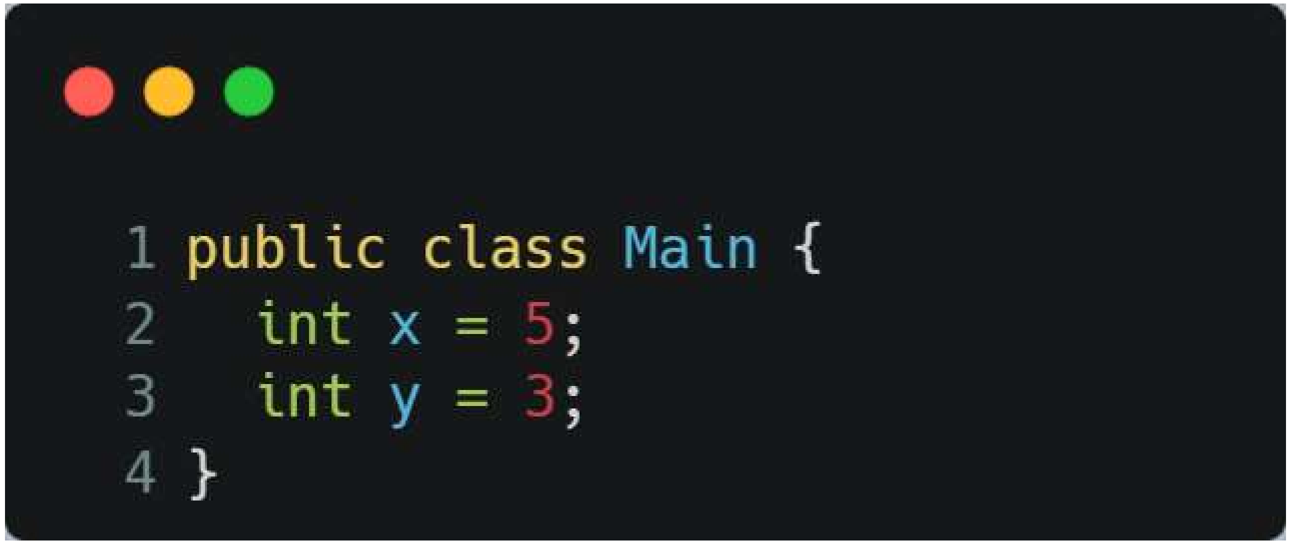
- Multiple Objects



```java
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

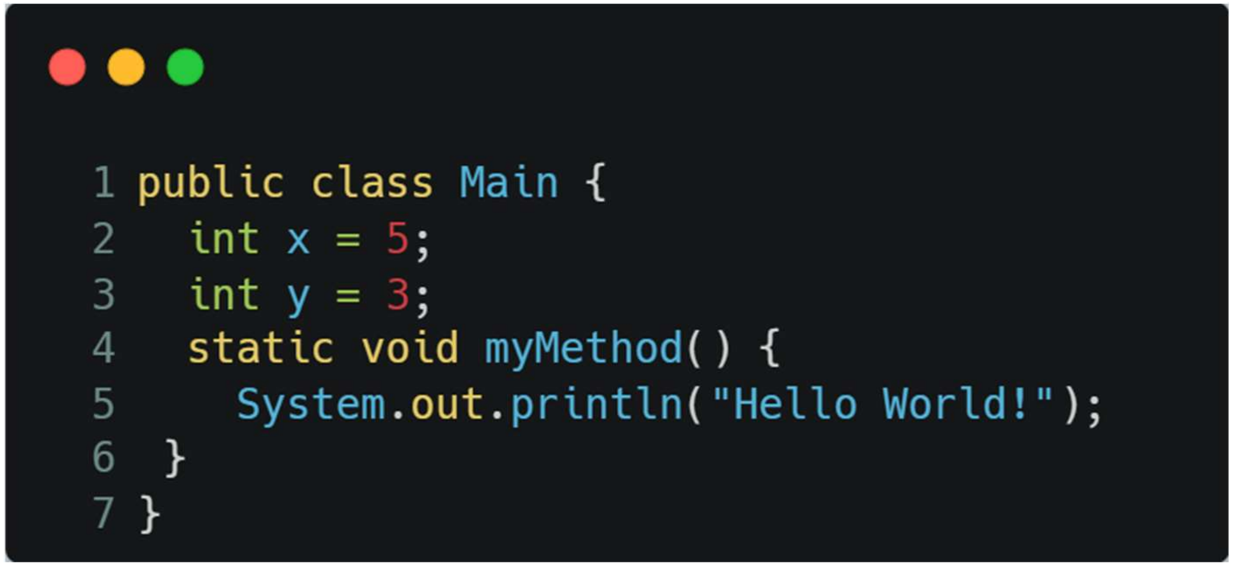
Class Attributes

- Java Class Attributes
  - In the previous chapter, we used the term "variable" for x in the example (as shown below). It is actually an attribute of the class. Or you could say that class attributes are variables within a class:

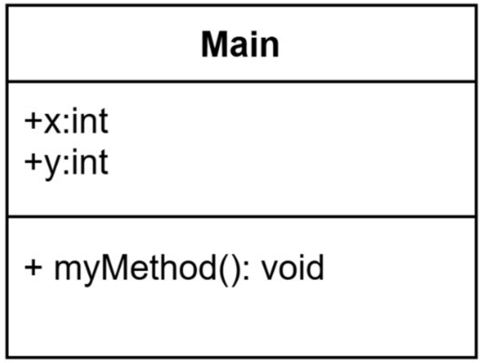


```java
public class Main {
  int x = 5;
  int y = 3;
}
```

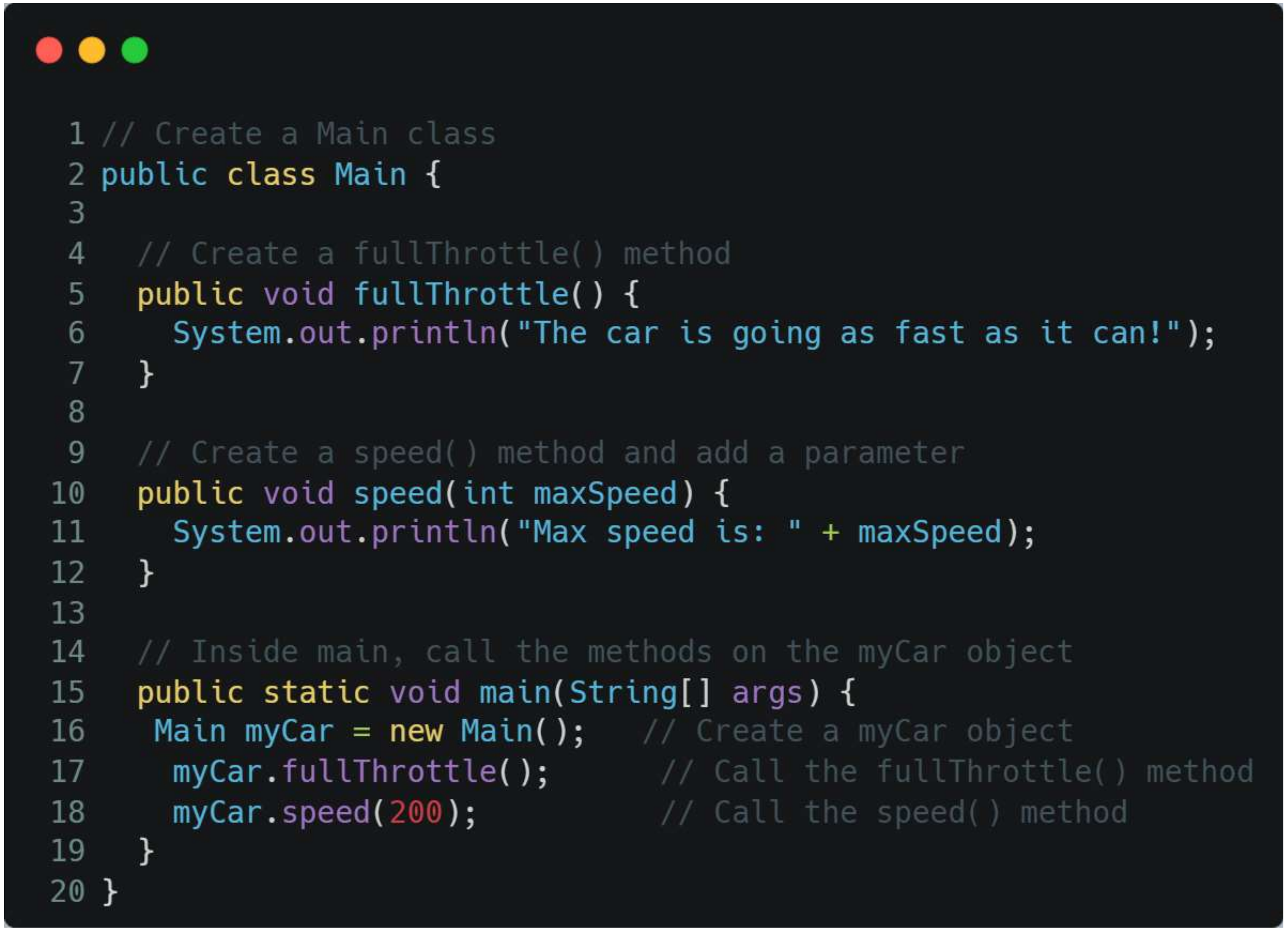
Attribute & Method



```java
public class Main {
  int x = 5;
  int y = 3;
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```



Attribute & Method



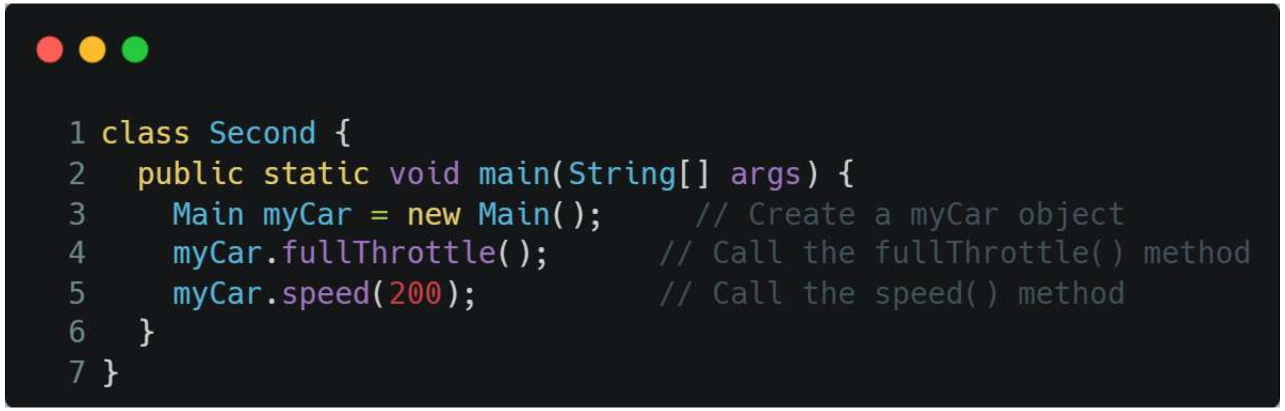
```java
// Create a Main class
public class Main {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
   Main myCar = new Main();   // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(200);          // Call the speed() method
  }
}
```

Attribute & Method

- Example explained
  1) We created a custom Main class with the class keyword.
  2) We created the fullThrottle() and speed() methods in the Main class.
  3) The fullThrottle() method and the speed() method will print out some text, when they are called.
  4) The speed() method accepts an int parameter called maxSpeed - we will use this in 8).
  5) In order to use the Main class and its methods, we need to create an object of the Main Class.
  6) Then, go to the main() method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).
  7) By using the new keyword we created an object with the name myCar.
  8) Then, we call the fullThrottle() and speed() methods on the myCar object, and run the program using the name of the object (myCar), followed by a dot (.), followed by the name of the method (fullThrottle(); and speed(200);). Notice that we add an int parameter of 200 inside the speed() method.

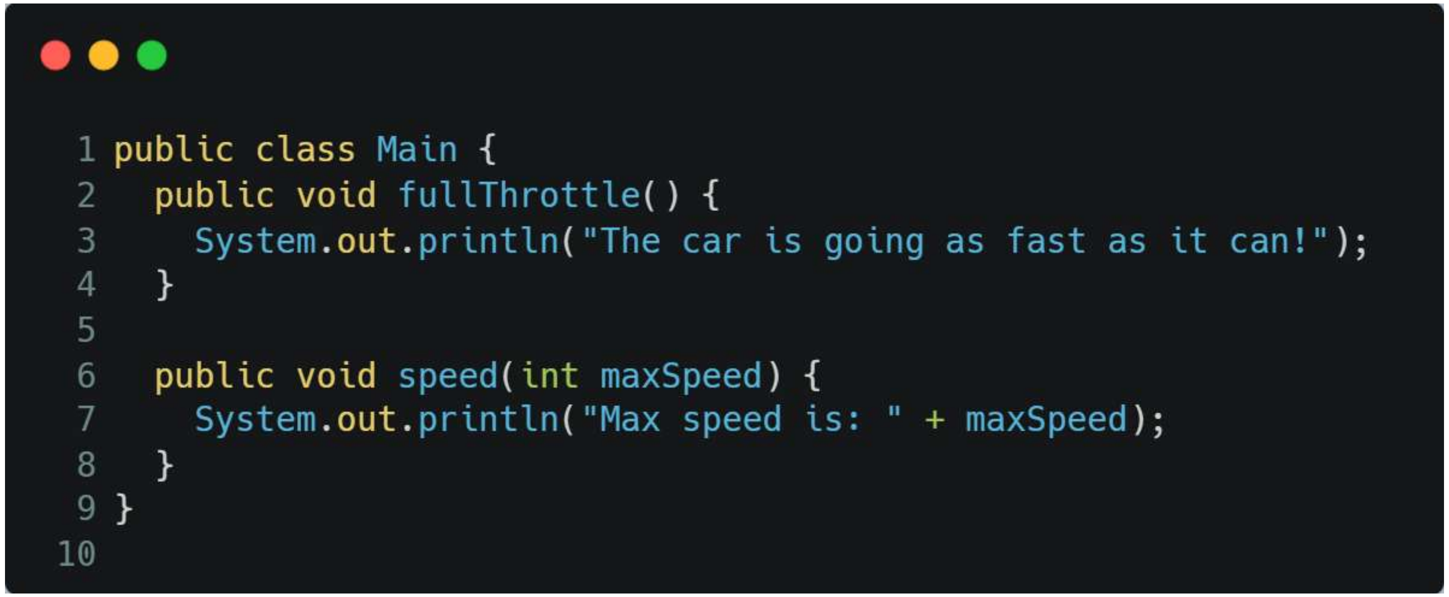
Using Multiple Classes

- Second.java



```java
class Second {
  public static void main(String[] args) {
    Main myCar = new Main();     // Create a myCar object
    myCar.fullThrottle();       // Call the fullThrottle() method
    myCar.speed(200);           // Call the speed() method
  }
}
```

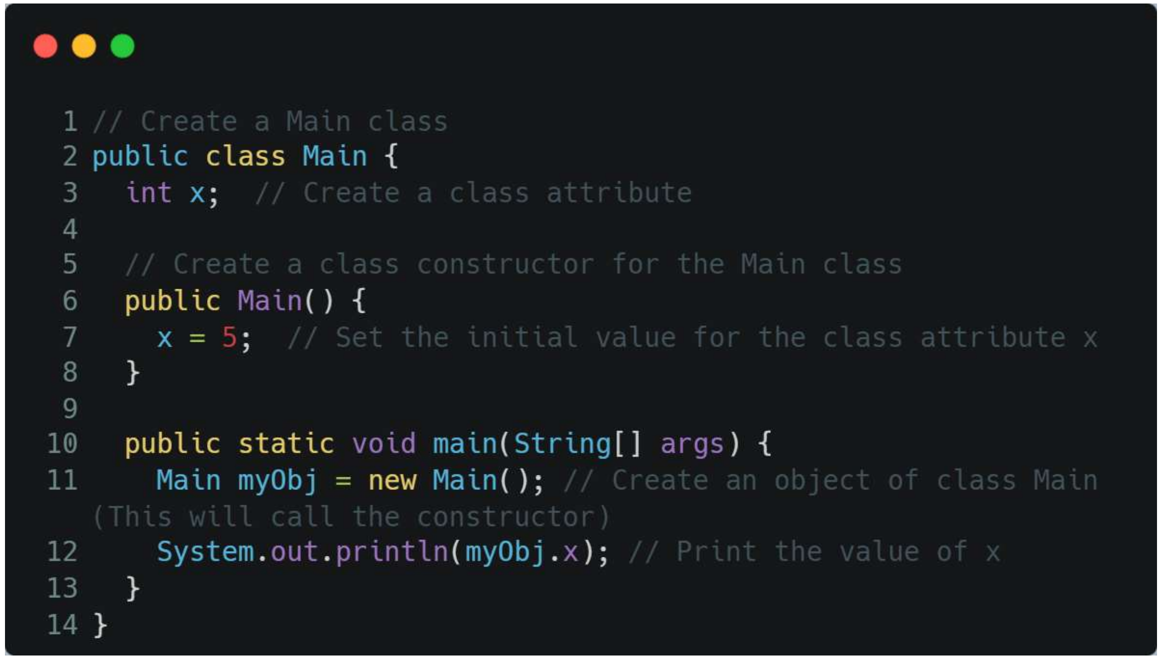
- Main.java



```java
public class Main {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

Java Constructors

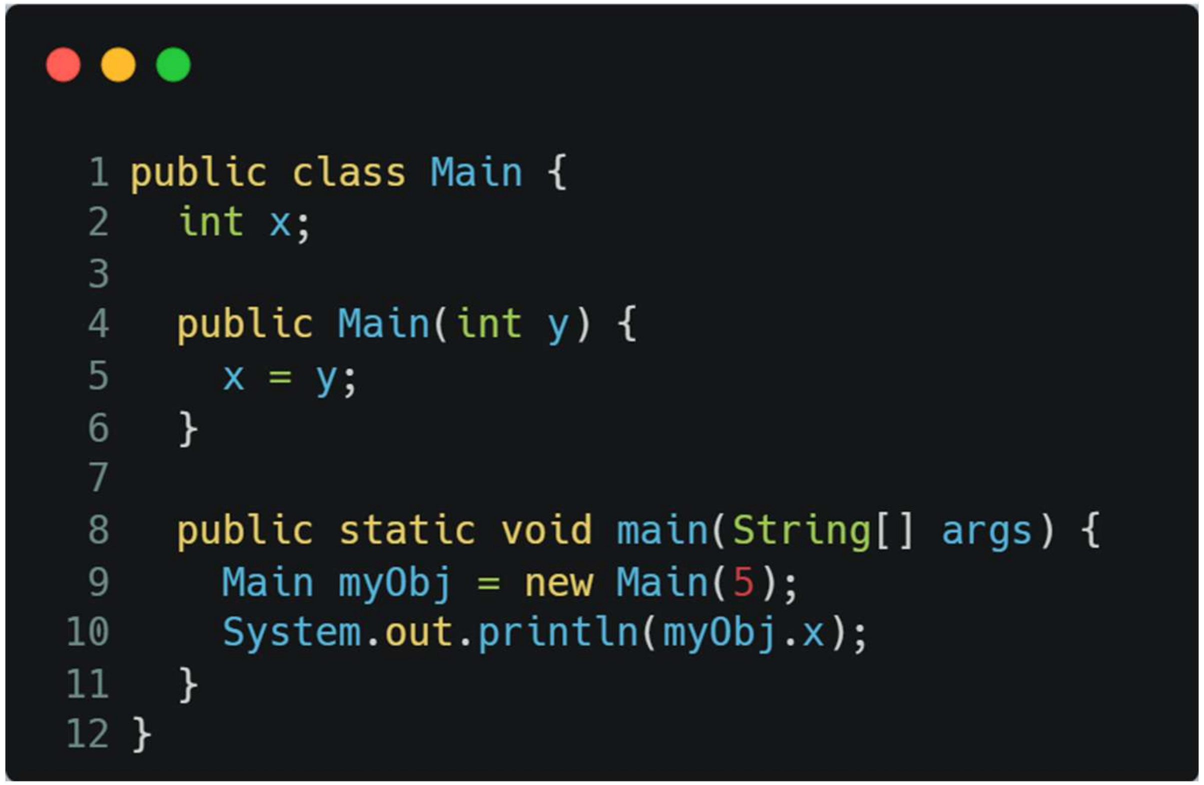
- A constructor in Java is a special method that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes:



```java
// Create a Main class
public class Main {
  int x;  // Create a class attribute

  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}
```

Constructors

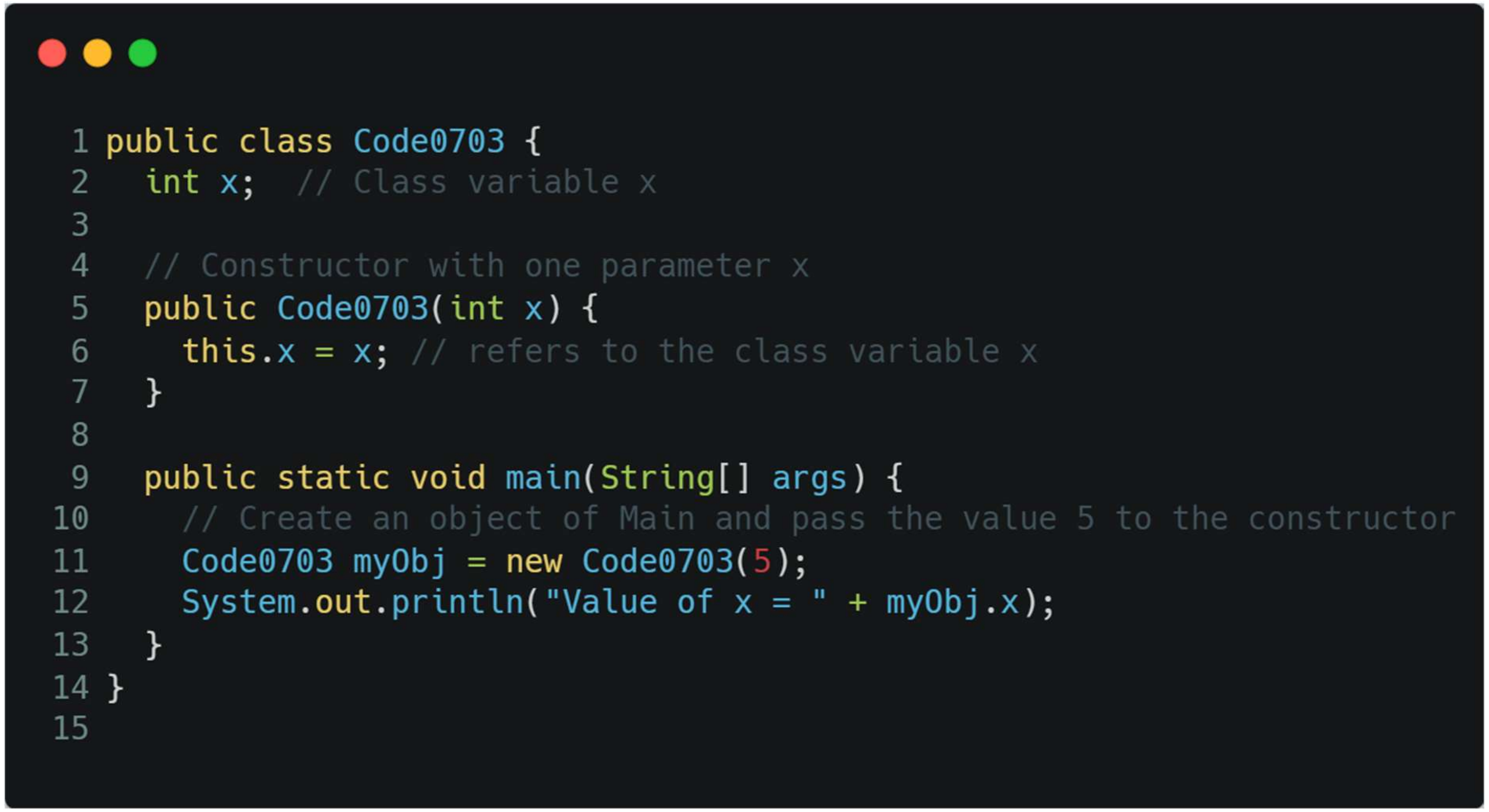
- Constructor Parameters
  - Constructors can also take parameters, which is used to initialize attributes.



```java
public class Main {
  int x;

  public Main(int y) {
    x = y;
  }

  public static void main(String[] args) {
    Main myObj = new Main(5);
    System.out.println(myObj.x);
  }
}
```

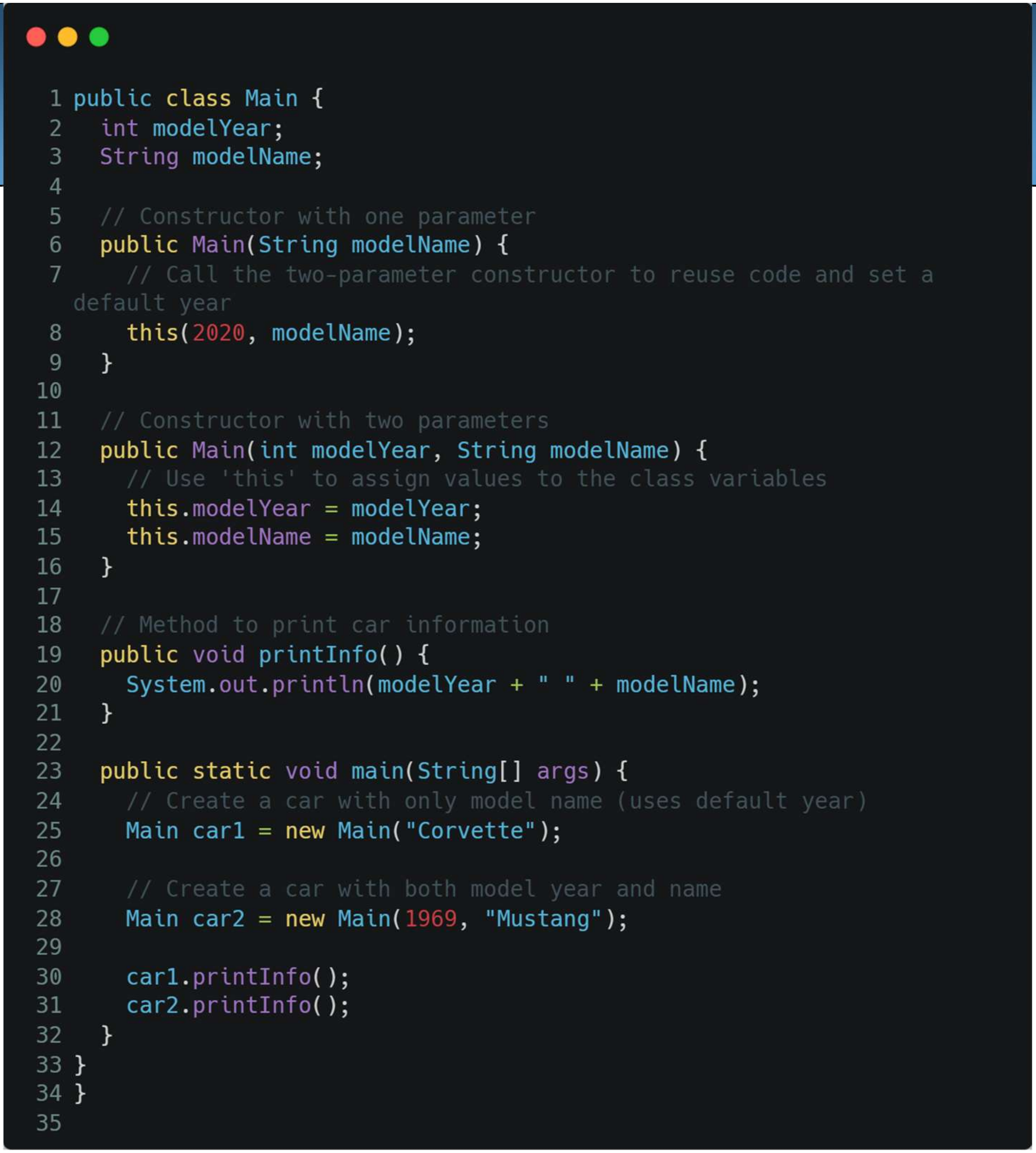
Multiple constructor

- The this keyword in Java refers to the current object in a method or constructor.



```java
public class Code0703 {
  int x;   // Class variable x

  // Constructor with one parameter x
  public Code0703(int x) {
    this.x = x; // refers to the class variable x
  }

  public static void main(String[] args) {
    // Create an object of Main and pass the value 5 to the constructor
    Code0703 myObj = new Code0703(5);
    System.out.println("Value of x = " + myObj.x);
  }
}

```

- Calling a Constructor from Another Constructor
  - You can also use this() to call another constructor in the same class.
  - This is useful when you want to provide default values or reuse initialization code instead of repeating it.



```java
public class Main {
  int modelYear;
  String modelName;

  // Constructor with one parameter
  public Main(String modelName) {
    // Call the two-parameter constructor to reuse code and set a default year
    this(2020, modelName);
  }

  // Constructor with two parameters
  public Main(int modelYear, String modelName) {
    // Use 'this' to assign values to the class variables
    this.modelYear = modelYear;
    this.modelName = modelName;
  }

  // Method to print car information
  public void printInfo() {
    System.out.println(modelYear + " " + modelName);
  }

  public static void main(String[] args) {
    // Create a car with only model name (uses default year)
    Main car1 = new Main("Corvette");

    // Create a car with both model year and name
    Main car2 = new Main(1969, "Mustang");

    car1.printInfo();
    car2.printInfo();
  }
}
}

```

Modifiers

- modifiers into two groups
  - Access Modifiers - controls the access level
  - Non-Access Modifiers - do not control access level, but provides other functionality

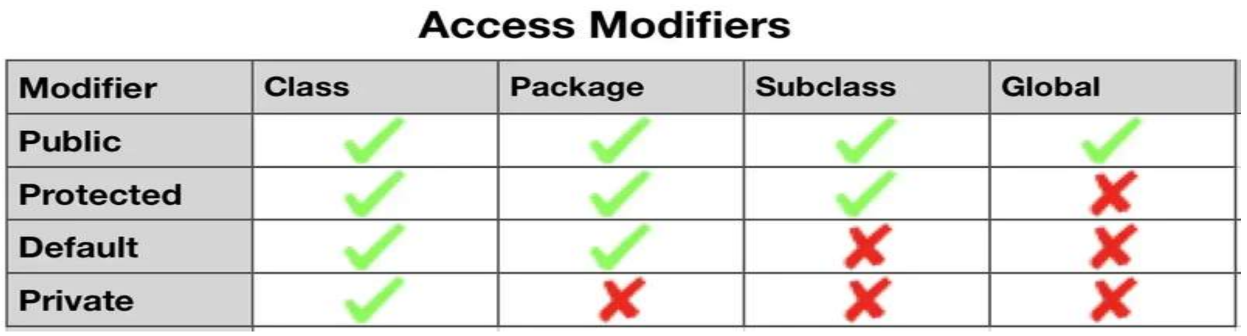
Access Modifiers

- For classes, you can use either public or default:
  - **public**
    - The class is accessible by any other class
  - *default*
    - The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter

Non-Access Modifiers

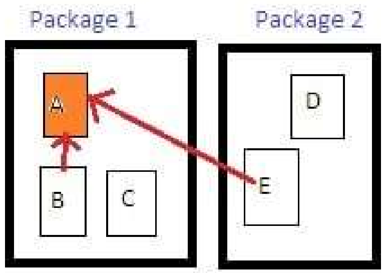
- attributes, methods and constructors
  - public
    - The code is accessible for all classes
  - private
    - The code is only accessible within the declared class
  - default
    - The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter
  - protected
    - The code is accessible in the same package and subclasses. You will learn more about subclasses and superclasses in the Inheritance chapter

Non-Access Modifiers

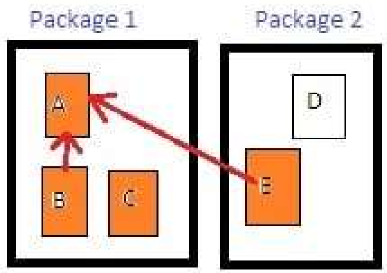


Access Modifiers

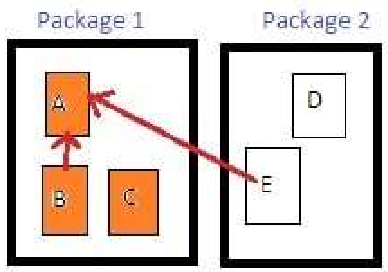
Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗



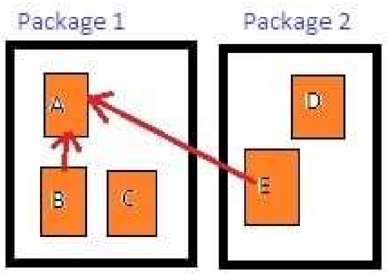
PRIVATE



PROTECTED

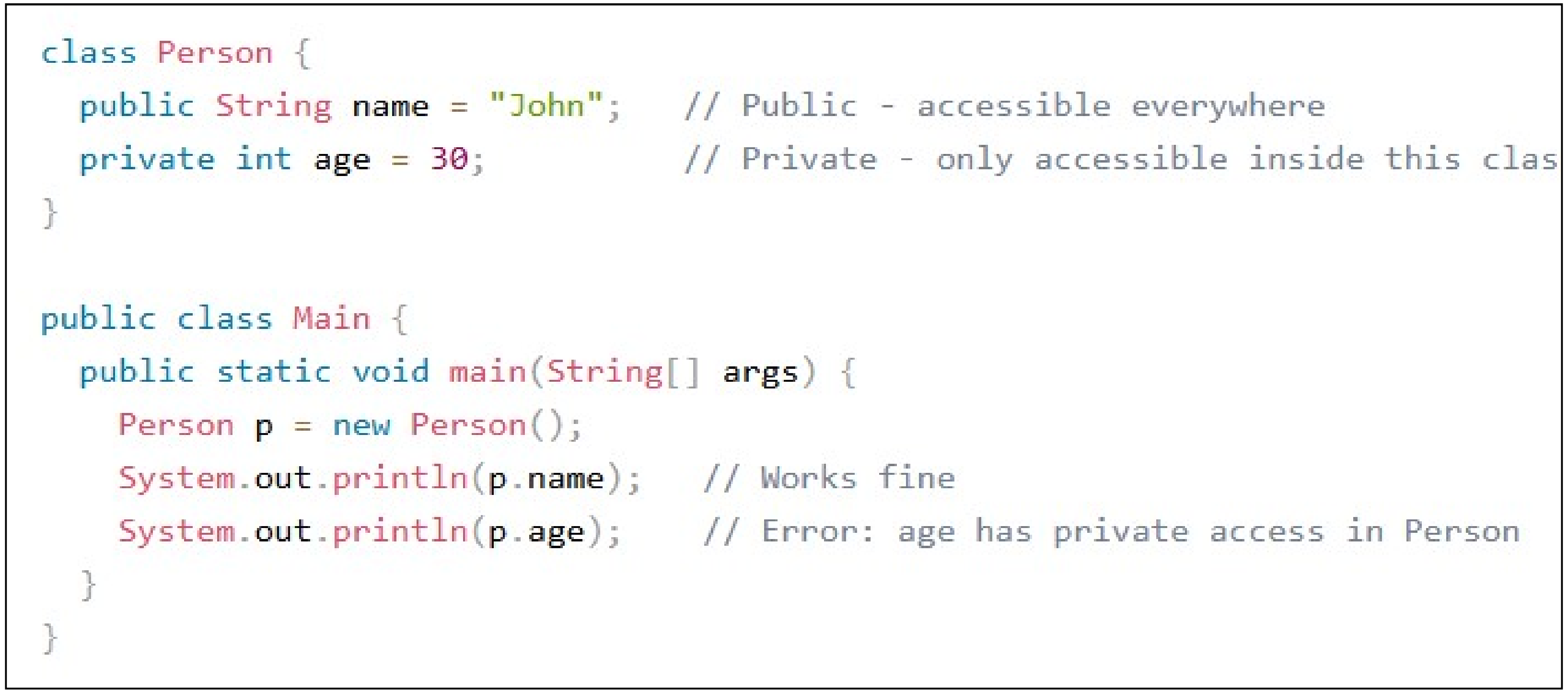


DEFAULT



PUBLIC

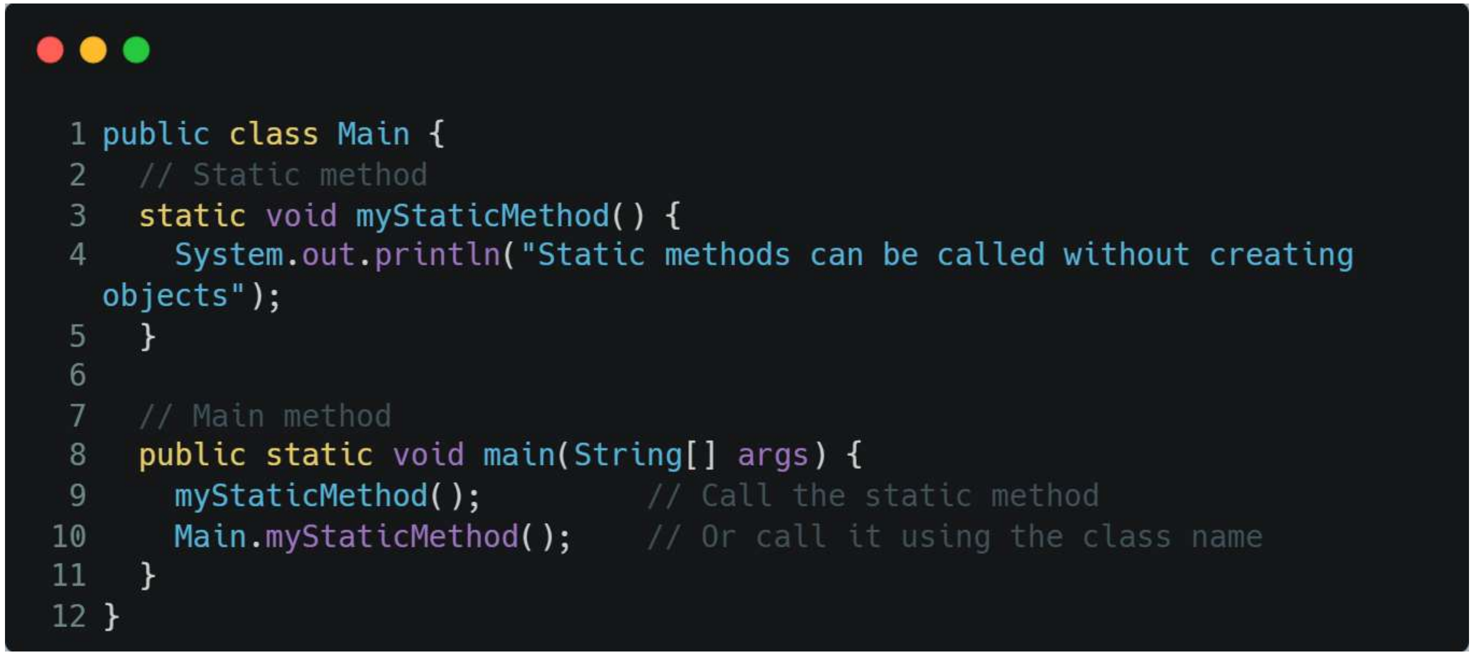
Non-Access Modifiers



```
class Person {
  public String name = "John";   // Public - accessible everywhere
  private int age = 30;          // Private - only accessible inside this clas
}

public class Main {
  public static void main(String[] args) {
    Person p = new Person();
    System.out.println(p.name);   // Works fine
    System.out.println(p.age);    // Error: age has private access in Person
  }
}
```

Non-Access Modifiers

- Static
  - A static method belongs to the class, not to any specific object.
  - This means you can call it without creating an object of the class.

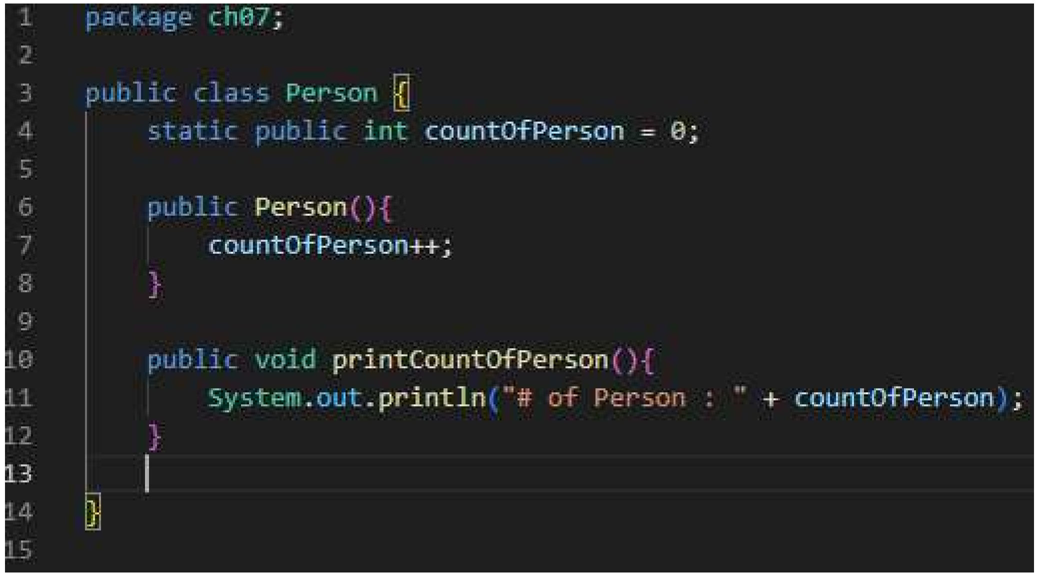


```java
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod();          // Call the static method
    Main.myStaticMethod();     // Or call it using the class name
  }
}
```

Non-Access Modifiers

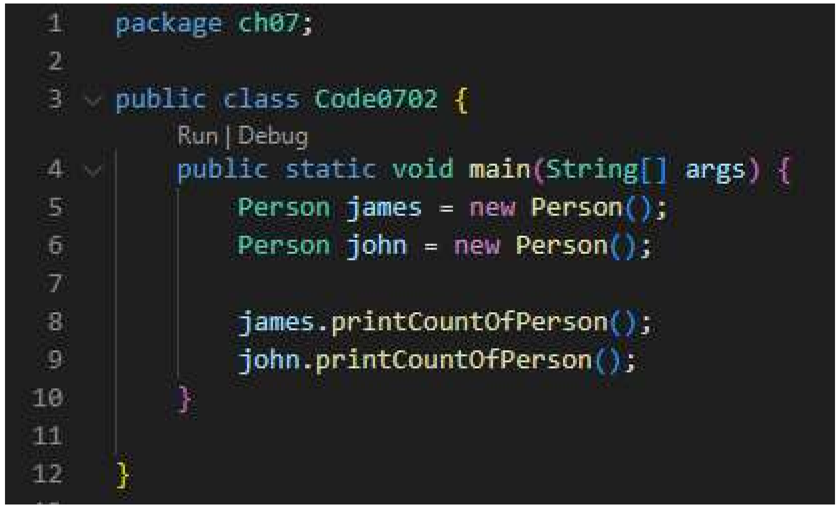
- Static

Person.java



```java
package ch07;

public class Person {
    static public int countOfPerson = 0;

    public Person(){
        countOfPerson++;
    }

    public void printCountOfPerson(){
        System.out.println("# of Person : " + countOfPerson);
    }

}
```

Code0702.java



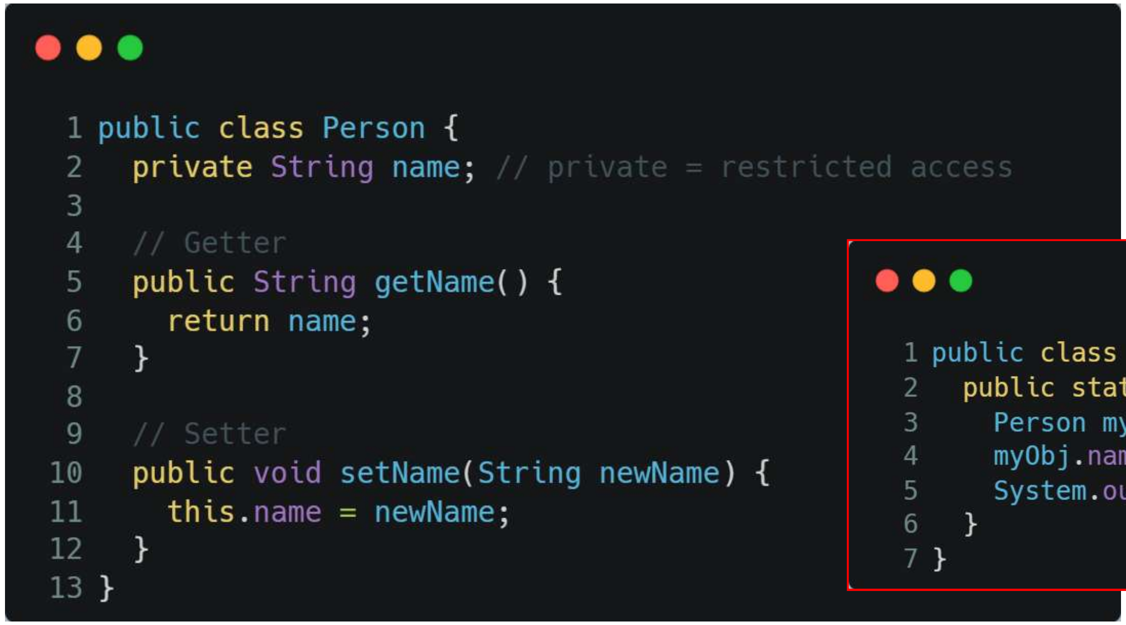
```java
package ch07;

public class Code0702 {
    Run | Debug
    public static void main(String[] args) {
        Person james = new Person();
        Person john = new Person();

        james.printCountOfPerson();
        john.printCountOfPerson();
    }

}
```

Encapsulation

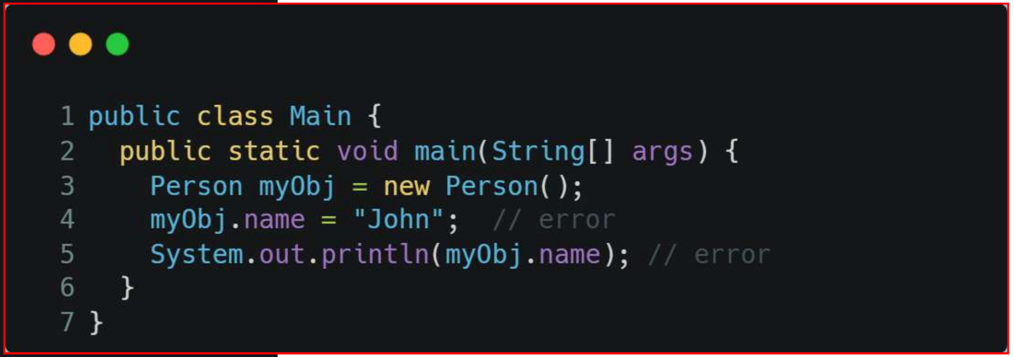
- Encapsulation
  - is to make sure that "sensitive" data is hidden from users.
  - To achieve this, you must:
    - declare class variables/attributes as private
    - provide public get and set methods to access and update the value of a private variable:
- Get and Set
  - private variables can only be accessed within the same class.
  - However, it is possible to access them if we provide public **get** and **set** methods.
  - The get method returns the variable value, and the set method sets the value.

Encapsulation

- getter / setter
  - The get method returns the value of the variable name.
  - The set method takes a parameter (newName) and assigns it to the name variable. The this keyword is used to refer to the current object.
  - However, as the name variable is declared as private, we cannot access it from outside this class:

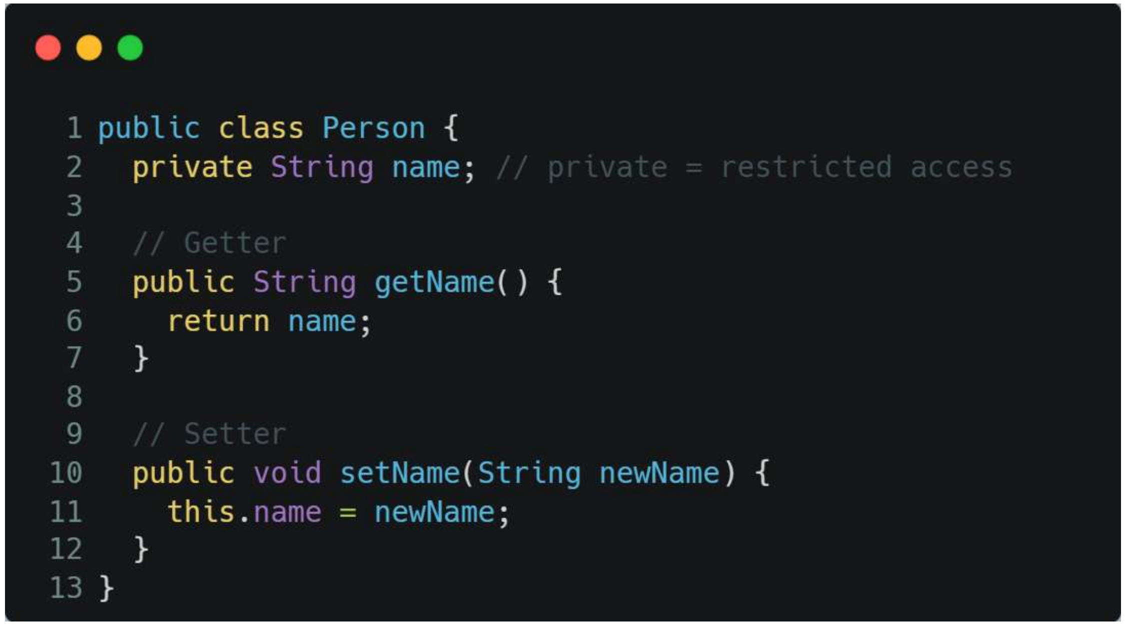


```java
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

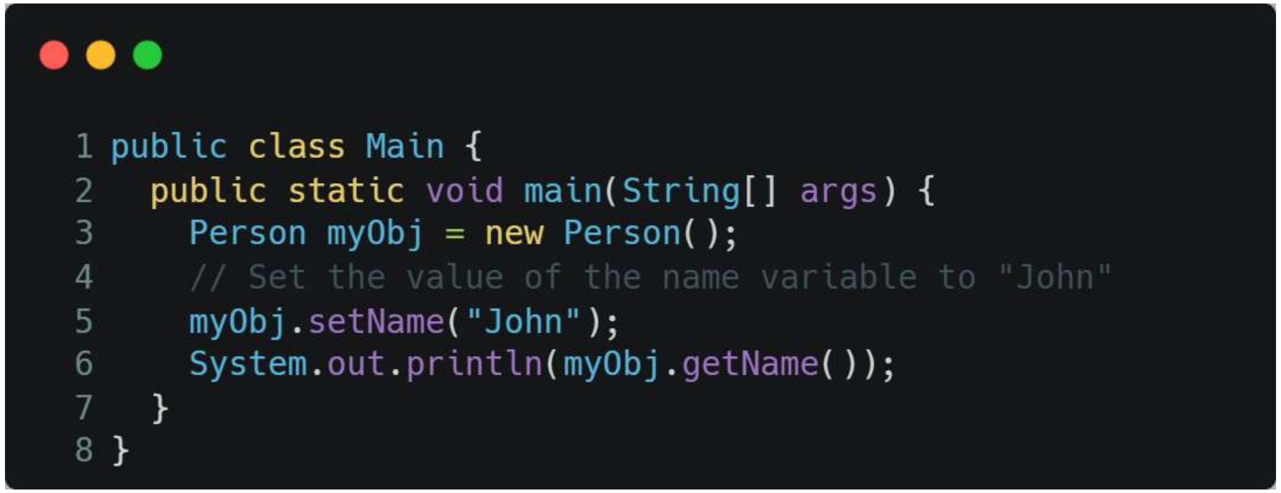


```java
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.name = "John";  // error
    System.out.println(myObj.name); // error
  }
}
```

Encapsulation



```java
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```



```java
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    // Set the value of the name variable to "John"
    myObj.setName("John");
    System.out.println(myObj.getName());
  }
}
```

Encapsulation

- Why Encapsulation?
  - Better control of class attributes and methods
  - Class attributes can be made read-only (if you only use the get method), or write-only (if you only use the set method)
  - Flexible: the programmer can change one part of the code without affecting other parts
  - Increased security of data

java Package

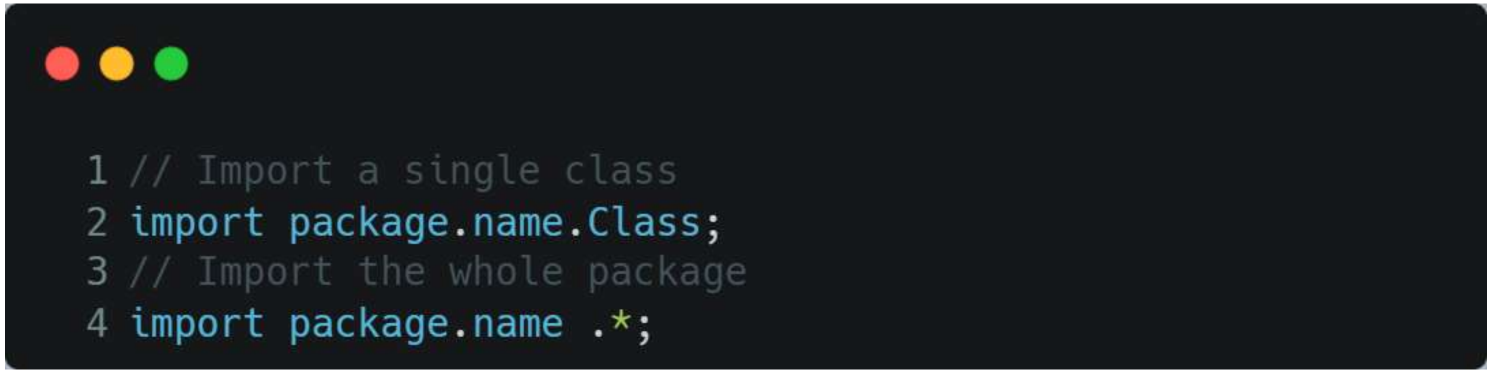
- A package in Java is used to group related classes.
- Think of it as a folder in a file directory.
- We use packages to avoid name conflicts, and to write a better maintainable code.
- Packages are divided into two categories:
  - Built-in Packages (packages from the Java API)
  - User-defined Packages (create your own packages)

Java Package

- Built-in Packages
  - The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.
  - The library contains components for managing input, database programming, and much much more. The complete list can be found at Oracles website: https://docs.oracle.com/en/java/javase/21/
  - The library is divided into **packages** and **classes**. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

Java Package

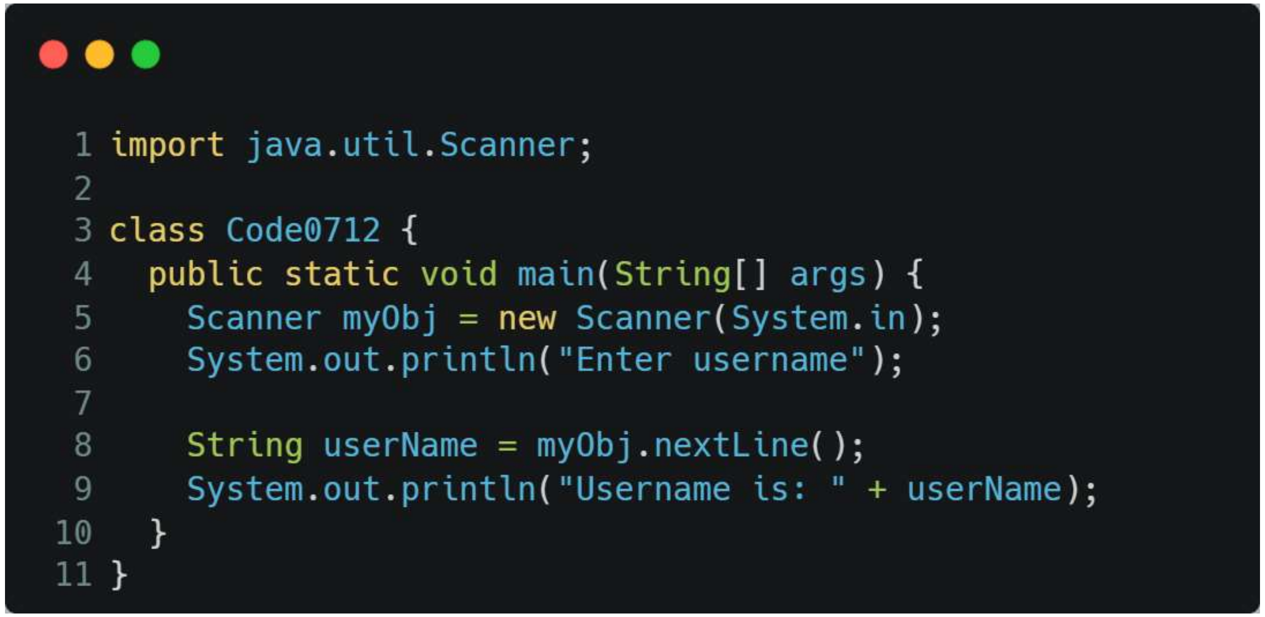
- To use a class or a package from the library, you need to use the *import* keyword:



```java
// Import a single class
import package.name.Class;
// Import the whole package
import package.name .*;
```

Java Package

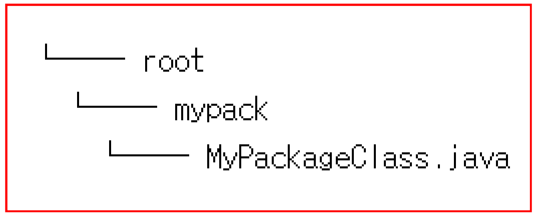
- In the example, java.util is a package, while Scanner is a class of the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read a complete line:



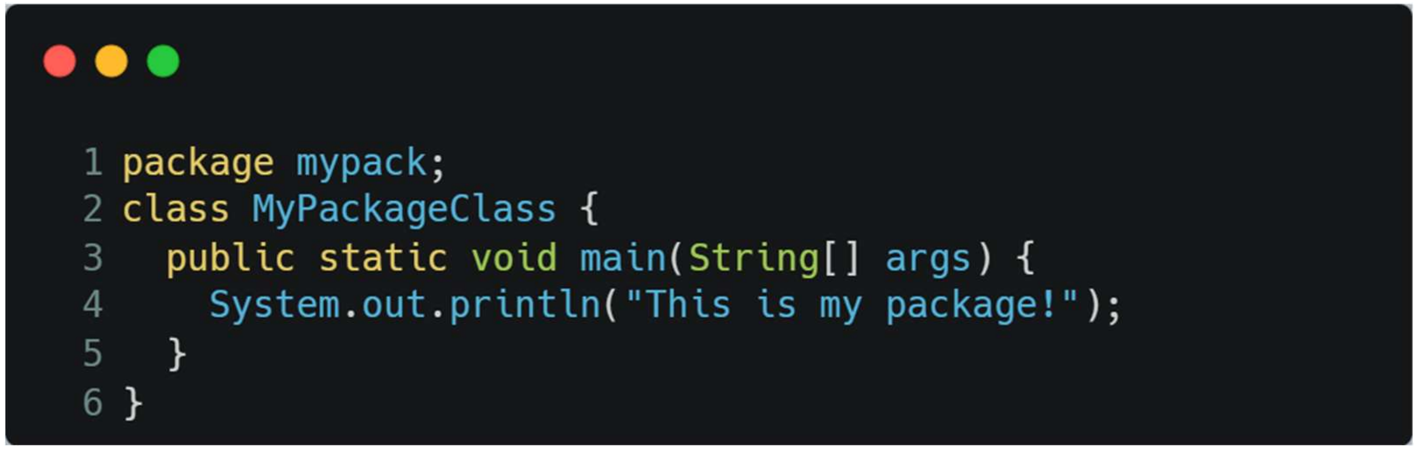
```java
import java.util.Scanner;

class Code0712 {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

User-defined Packages

- To create your own package, you need to understand that Java uses a file system directory to store them.
- Just like folders on your computer:



```
└──── root
  └──── mypack
    └──── MyPackageClass.java
```



```java
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```