Java Bootcamp

Session #3

The DA Programming Club

Object Oriented Programming (OOP)

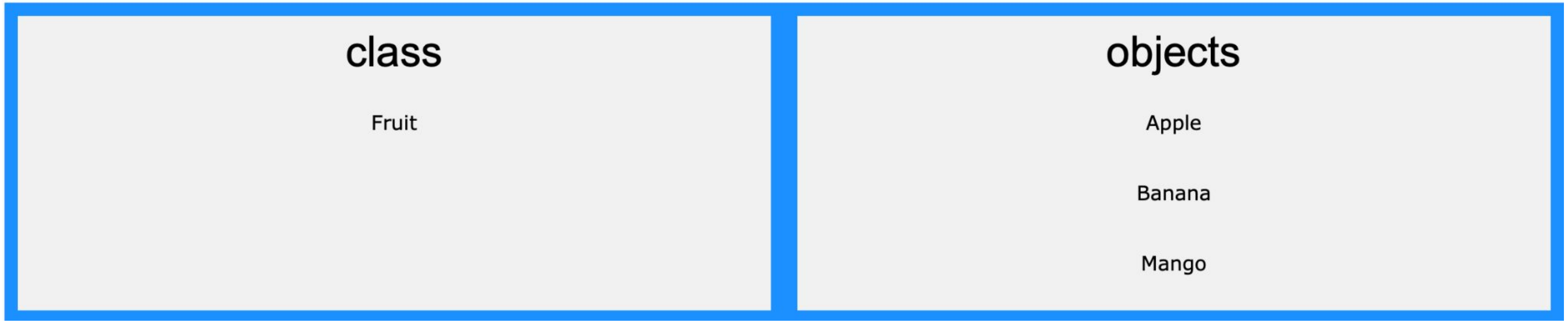
Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

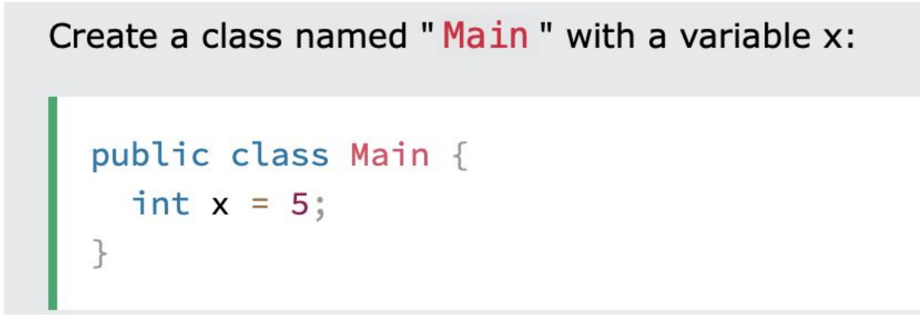
- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development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

Classes VS Objects

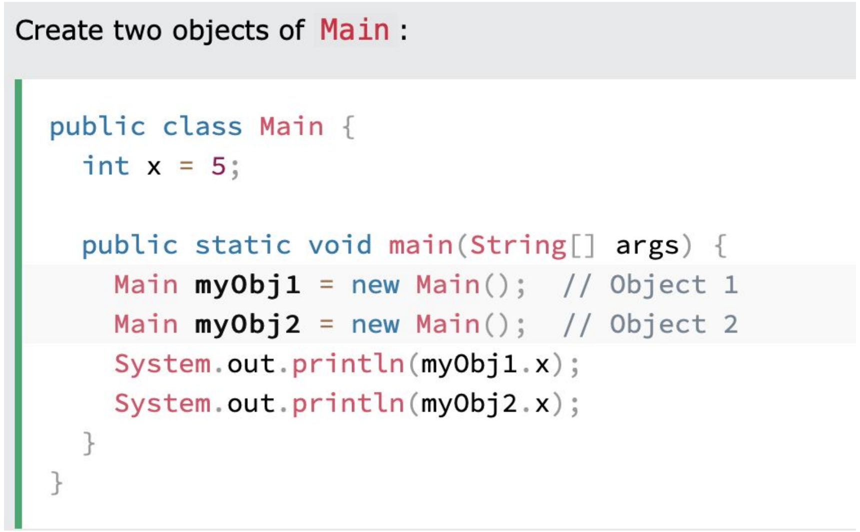
A class is a template for objects, and an object is an instance of a class. When the individual objects are created, they inherit all the variables and methods from the class.



Creating a Class and Instantiating It


Create a class named "Main" with a variable x:

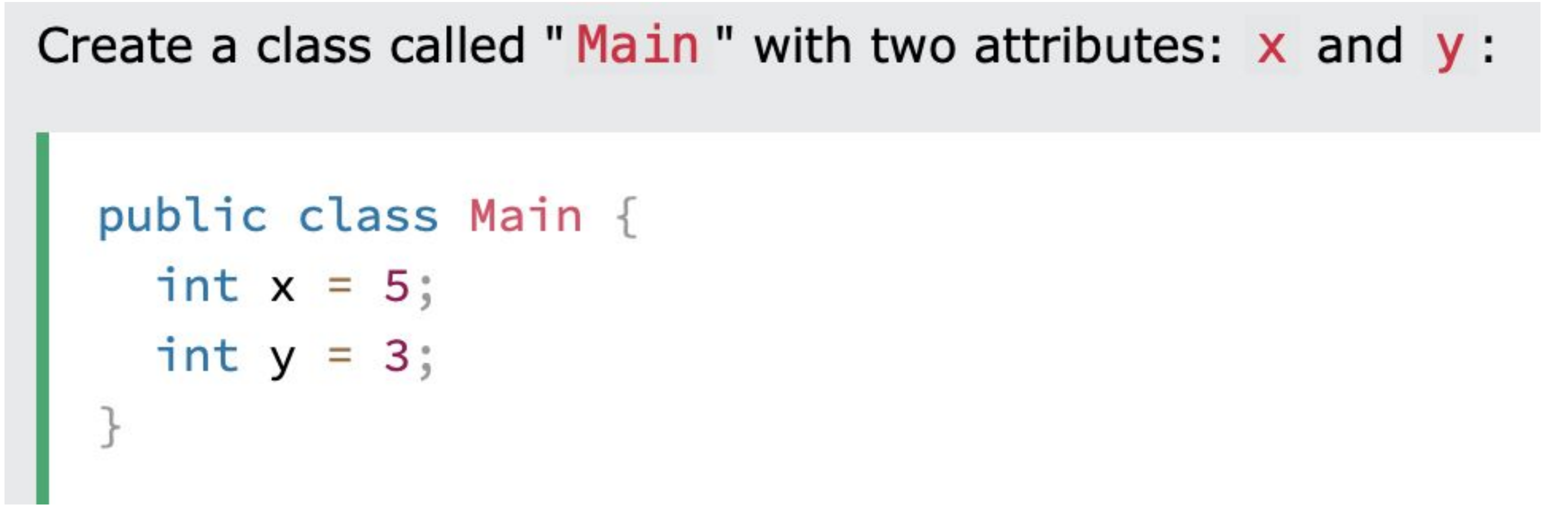
```
public class Main {
  int x = 5;
}
```


Create two objects of Main :

```
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);
  }
}
```

Attributes

Any variable that is bound in a class is a class attribute.

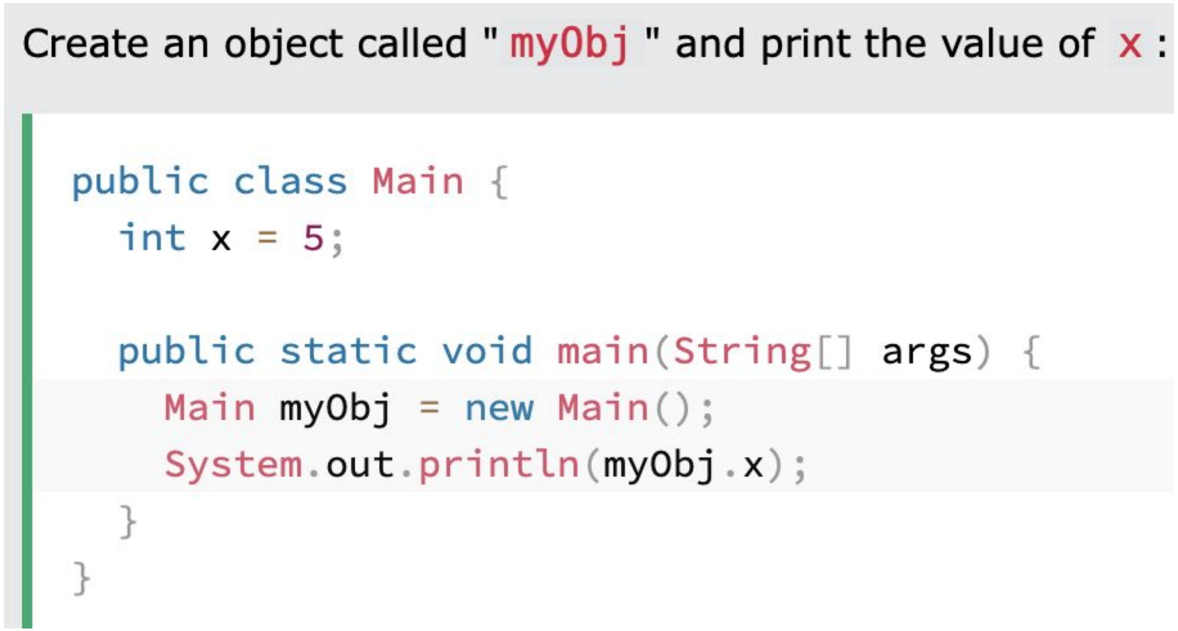


Create a class called "Main" with two attributes: x and y:

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

Attributes

You can access attributes by creating an object of the class, and by using the dot syntax (.):

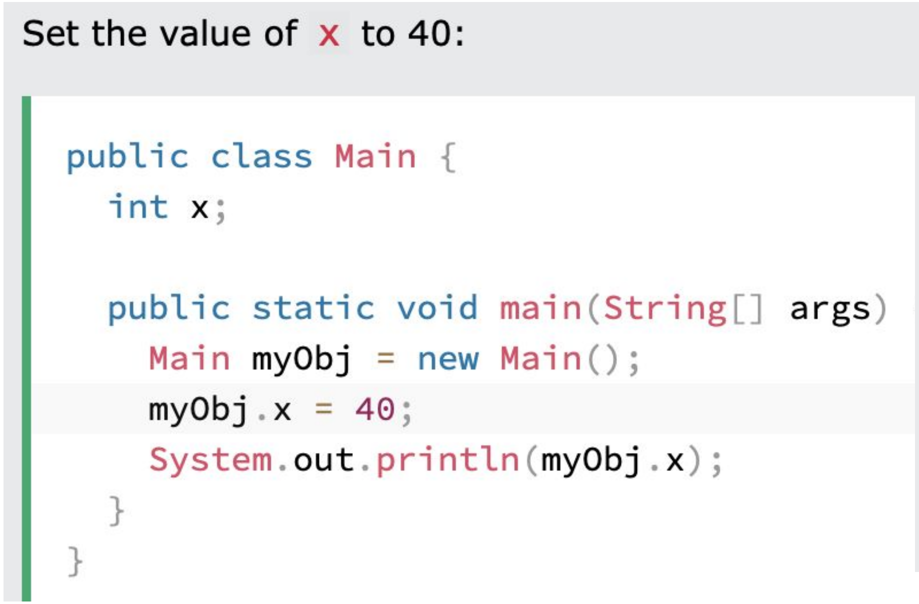


Create an object called "myObj" and print the value of x:

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

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

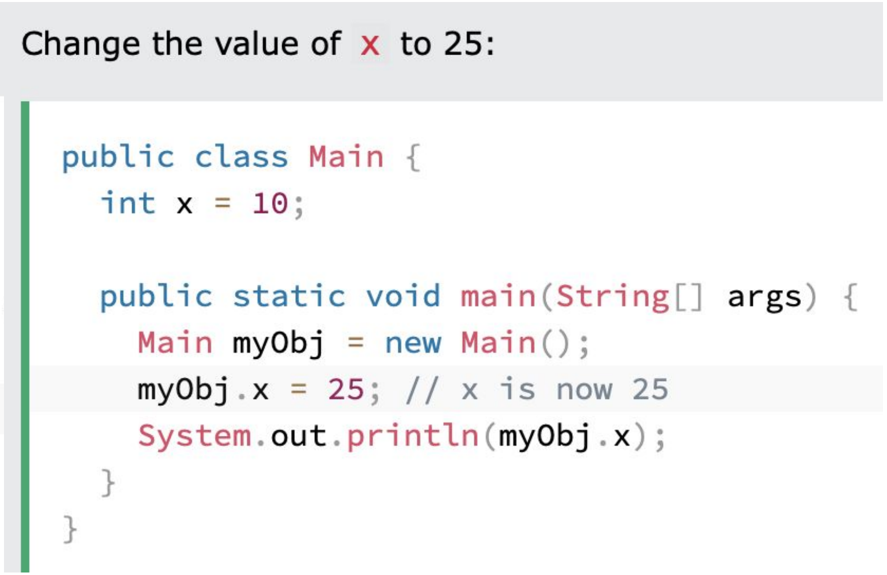
Attributes

You can modify attribute values or override existing ones:



Set the value of `x` to 40:

```java
public class Main {
  int x;

  public static void main(String[] args)
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

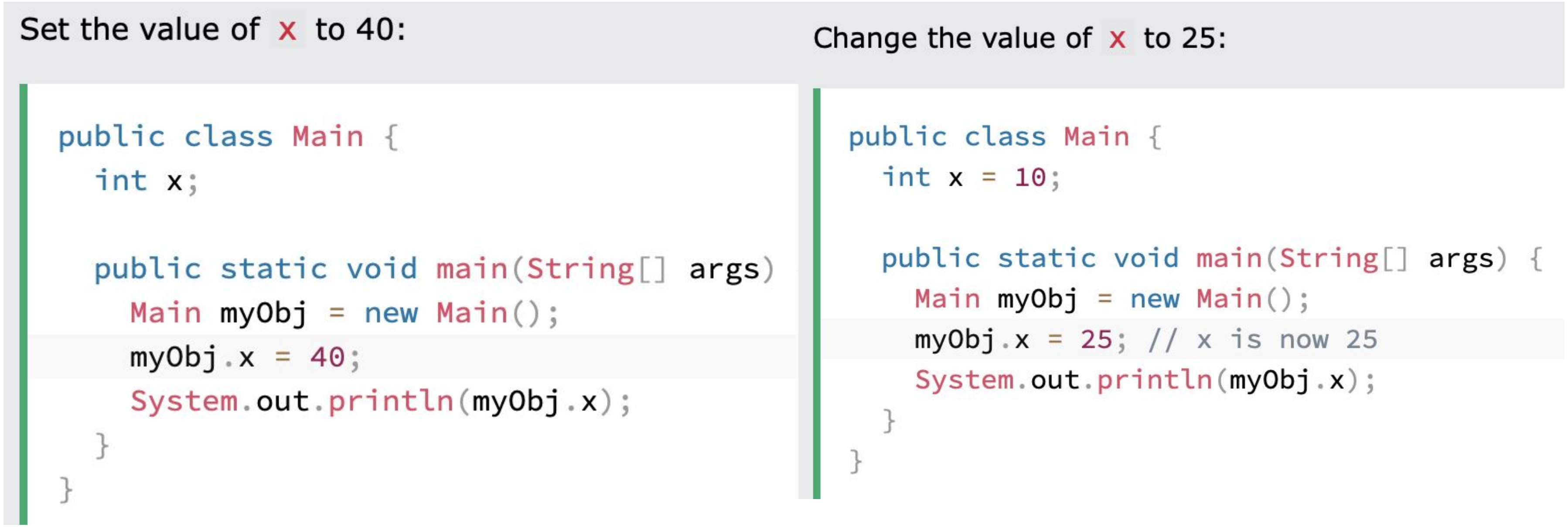


Change the value of `x` to 25:

```java
public class Main {
  int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

Attributes

If you don't want the ability to override existing values, declare the attribute as `final`:



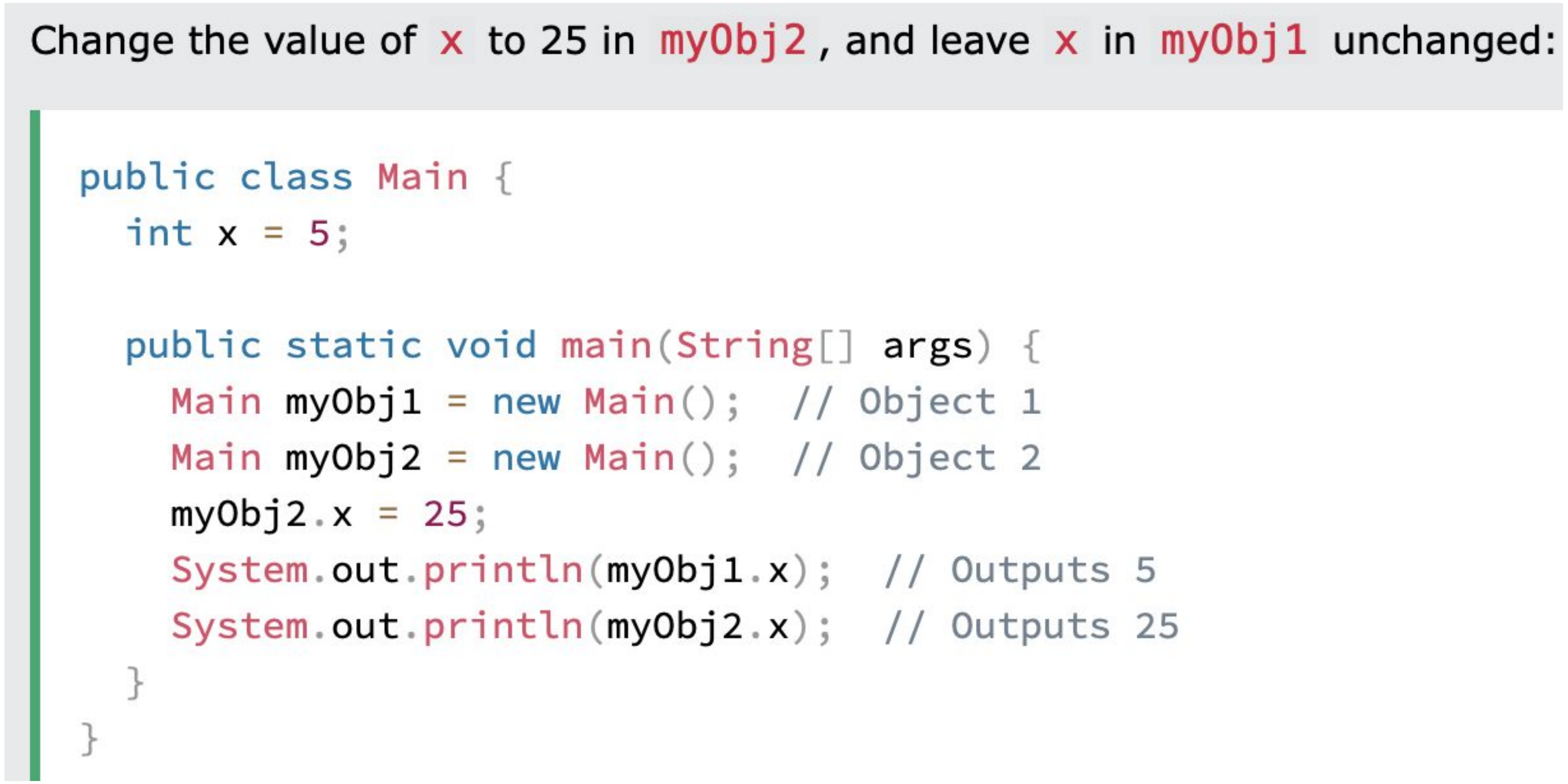
Set the value of `x` to 40:

```java
public class Main {
  int x;

  public static void main(String[] args)
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

Change the value of `x` to 25:

```java
public class Main {
  int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

Attributes

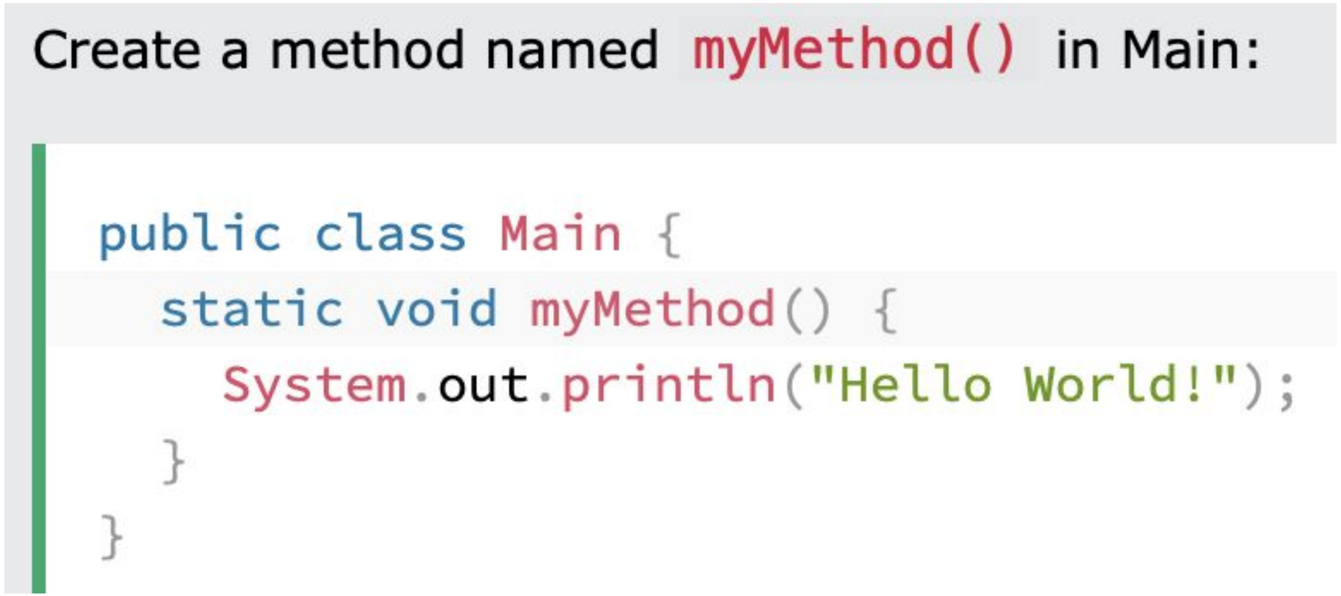


Change the value of x to 25 in myObj2, and leave x in myObj1 unchanged:

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

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Methods

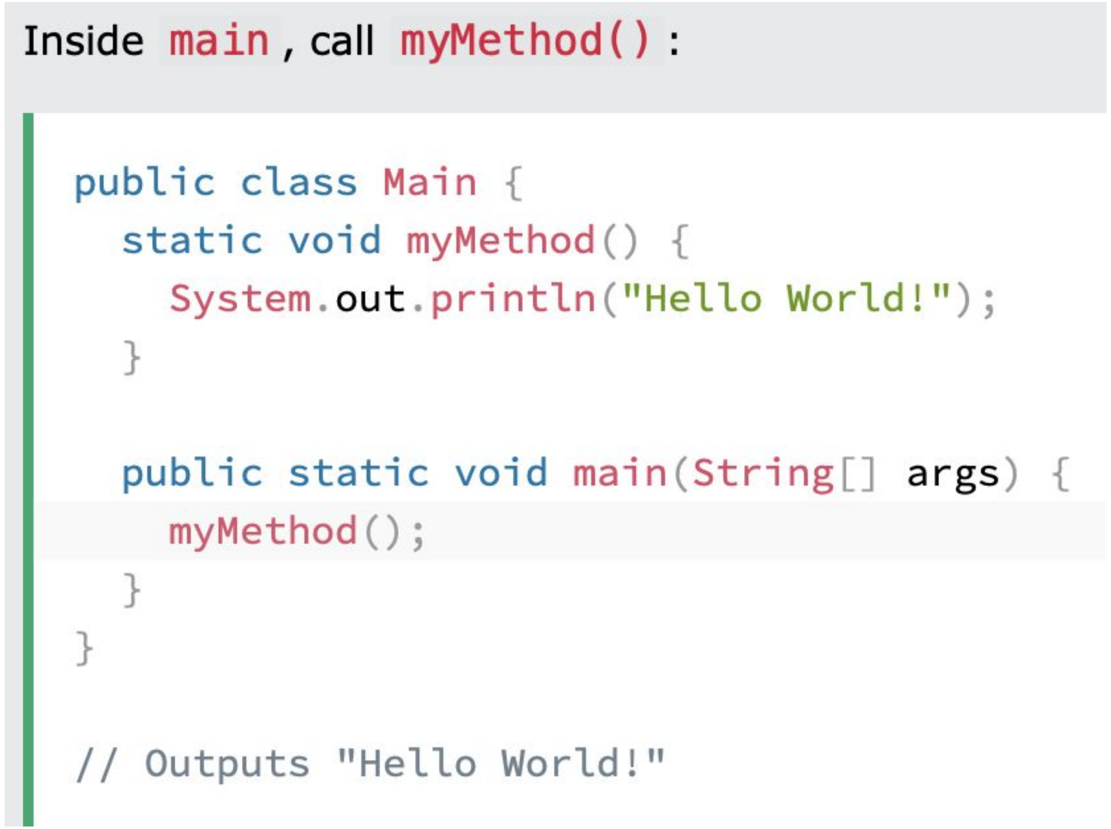
Any function defined within a class is a method.



Create a method named `myMethod()` in Main:

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

Methods



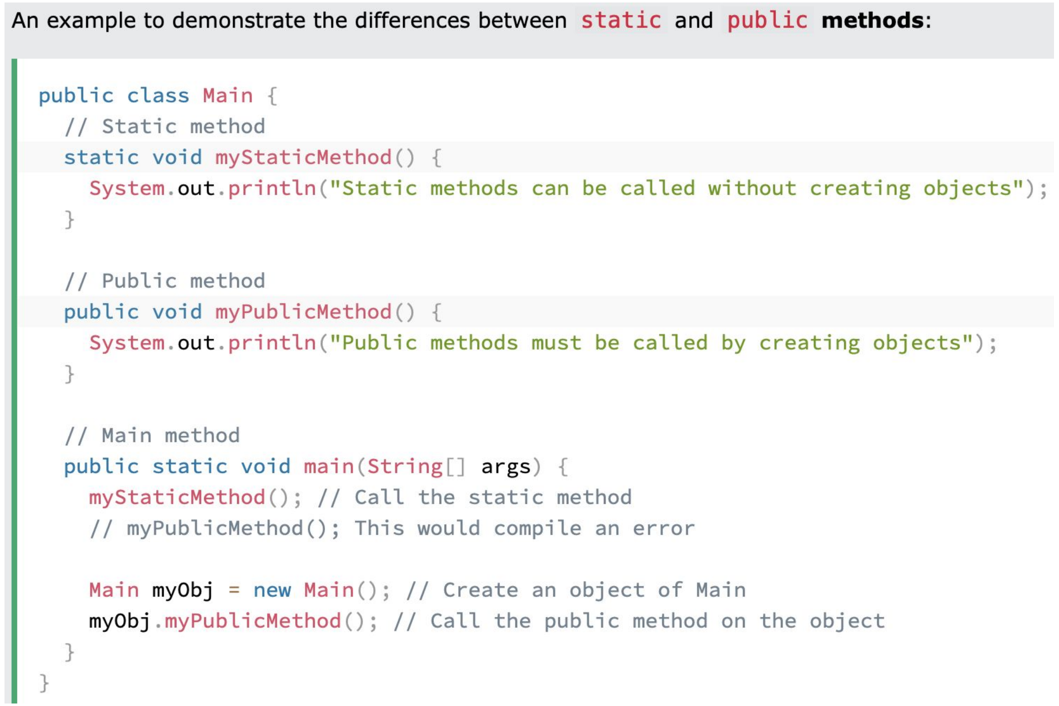
Inside `main`, call `myMethod()`:

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

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

// Outputs "Hello World!"
```

Methods

You will often see Java programs that have either static or public attributes and methods.

In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:


An example to demonstrate the differences between `static` and `public` **methods**:

```
public class Main {
  // 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

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

Create a Car object named `myCar`. Call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program:

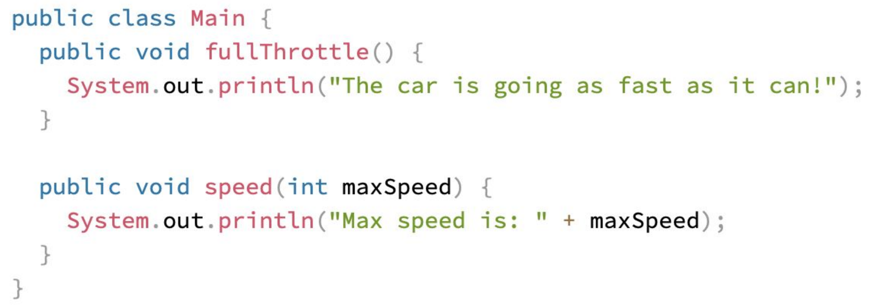
```java
// Create a Main class
public class Main {

  // 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
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```

Good Practice

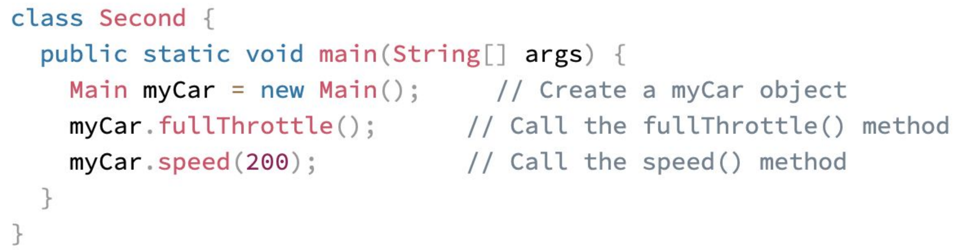
It is a good practice to create an object of a class and access it in another class.

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory:

- Main.java
- Second.java



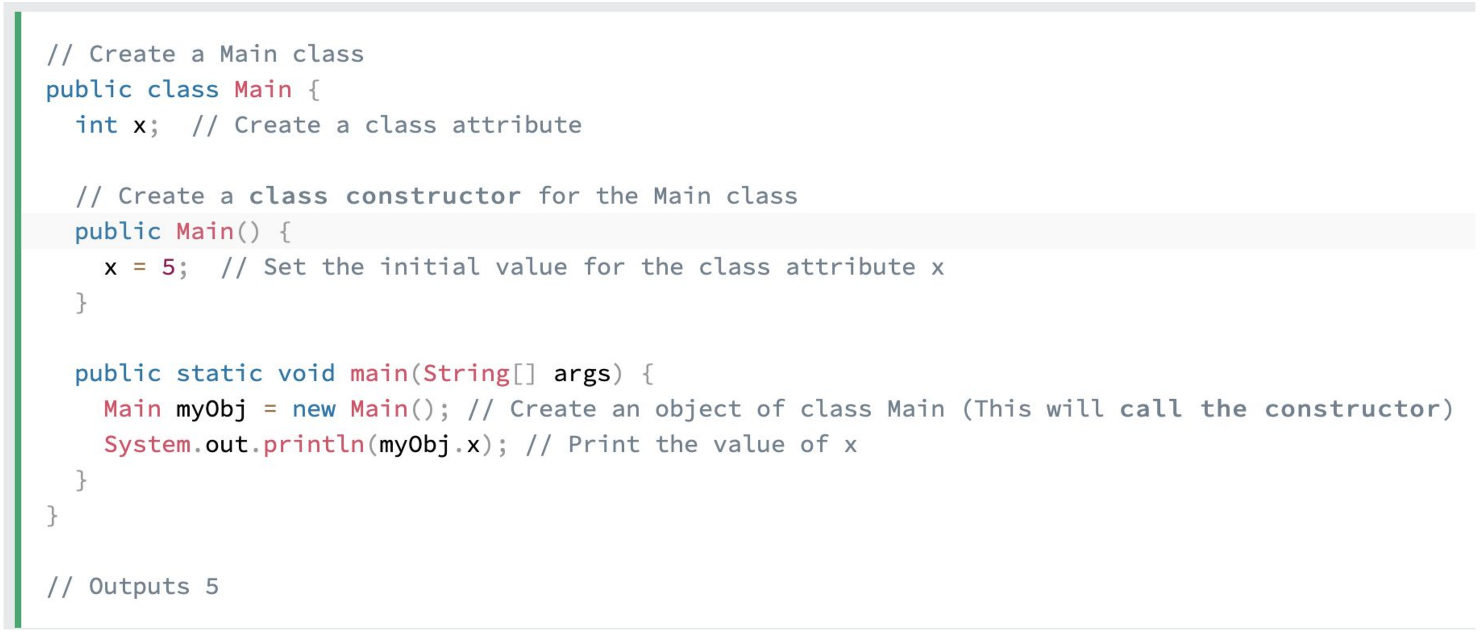
```java
public class Main {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

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



```java
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
  }
}
```

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
  }
}

// Outputs 5
```

Constructors: Important to know

1. Note that the constructor name must match the class name, and it cannot have a return type (like void).
2. Also note that the constructor is called when the object is created.
3. All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

Constructors

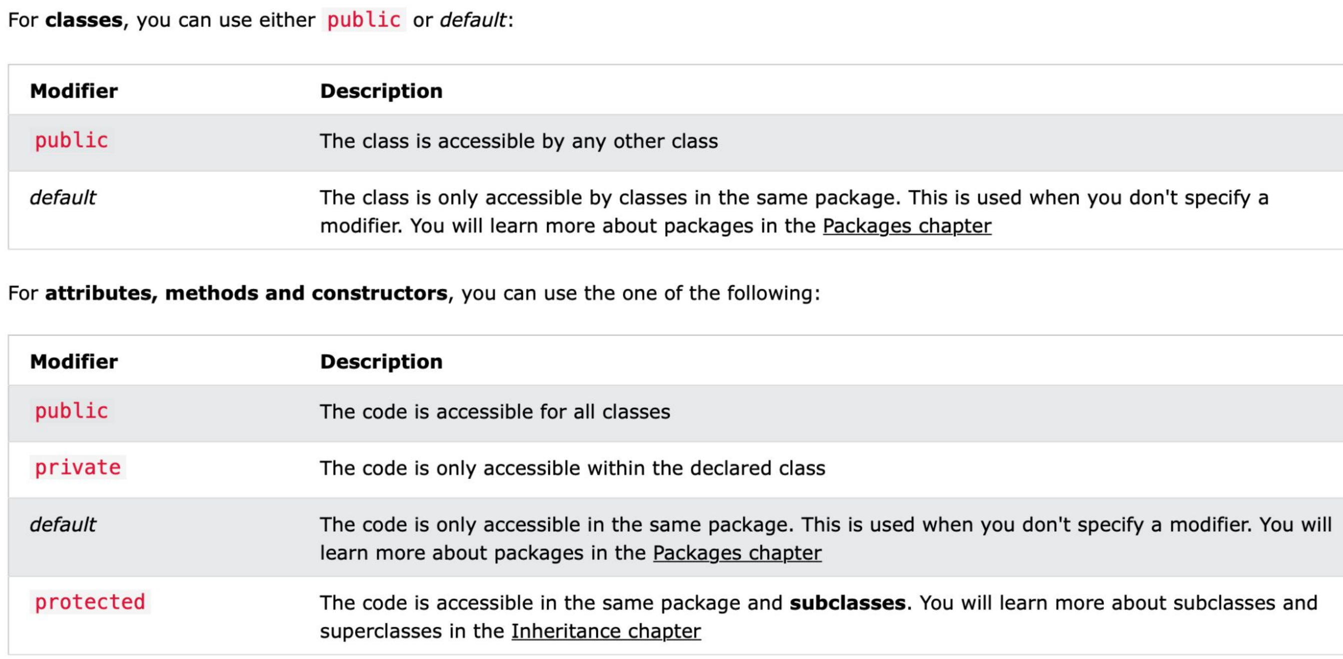
```
public class Main {
  int modelYear;
  String modelName;

  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}

// Outputs 1969 Mustang
```

Modifiers

We divide modifiers into two groups:

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

Non-access modifiers available here



For **classes**, you can use either `public` or *default*:

Modifier	Description
`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

For **attributes, methods and constructors**, you can use the one of the following:

Modifier	Description
`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

Encapsulation

The meaning of 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

Why does encapsulation matter?

- 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

Encapsulation

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

```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.setName("John"); // Set the value of the name variable to "John"
    System.out.println(myObj.getName());
  }
}

// Outputs "John"
```

Try it out!

Start with the two Java classes available on Blackboard, fill in the TODO comment lines, and print the output!