CJ Information

S – AP CS A

IC – Getter & Setter

HW – Exercises at the end of the topic

A – 2024.11.07 Thu – TEST

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Outline :

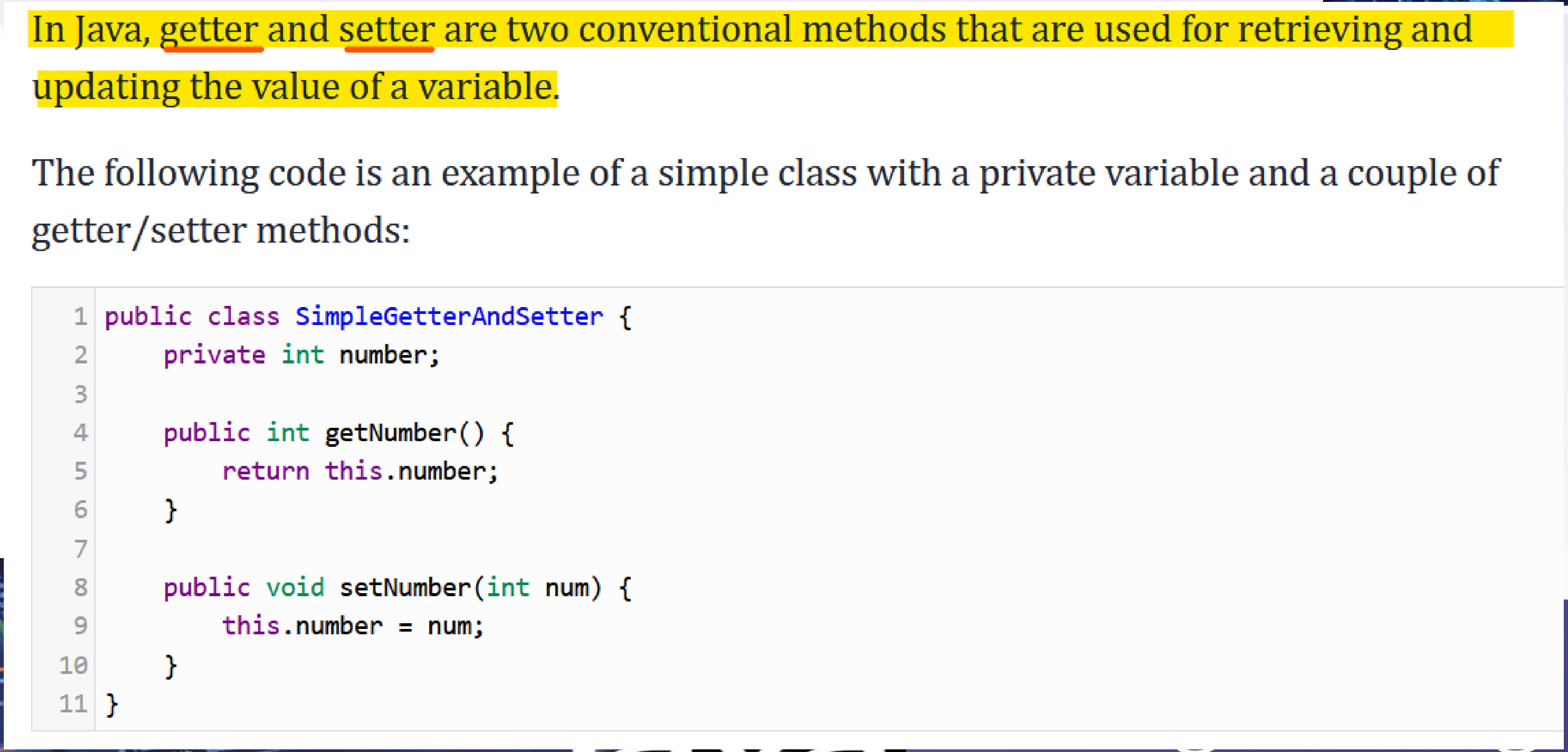
- What Getter & Setter are ?
- Why do we need them in our Java code?
- How to implement them?

What we learn today:

- How to use Getter & Setter
- Common Mistakes
- Example

Getters & Setters

What are Getter & Setter?



In Java, getter and setter are two conventional methods that are used for retrieving and updating the value of a variable.

The following code is an example of a simple class with a private variable and a couple of getter/setter methods:

```java
public class SimpleGetterAndSetter {
    private int number;

    public int getNumber() {
        return this.number;
    }

    public void setNumber(int num) {
        this.number = num;
    }
}
```

Getters & Setters

What are Getter & Setter?

The class declares a private variable, number. Since number is private, the code from the outside of this class cannot access the variable directly, as shown below:

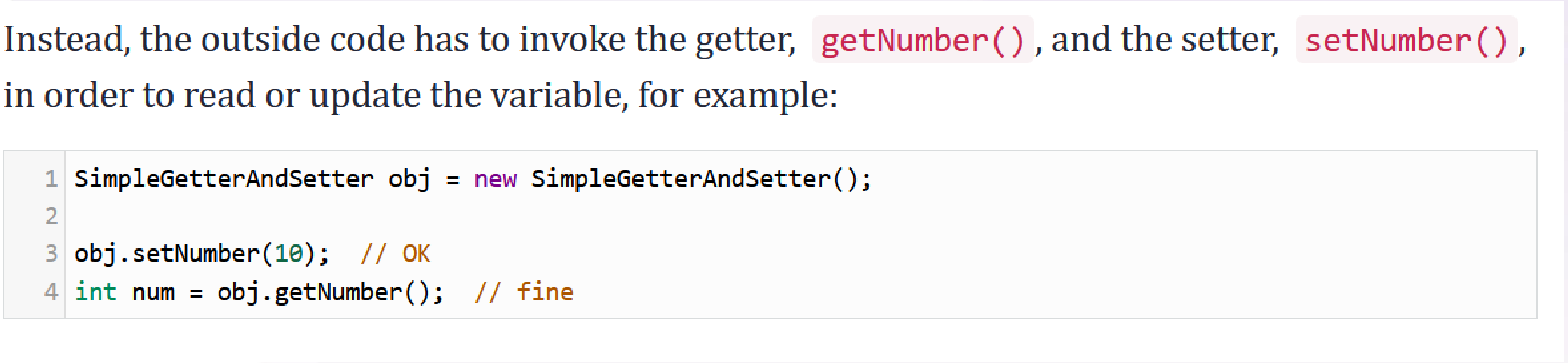
```
SimpleGetterAndSetter obj = new SimpleGetterAndSetter();
obj.number = 10;    // compile error, since number is private
int num = obj.number; // same as above
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

What are Getter & Setter?



Instead, the outside code has to invoke the getter, `getNumber()`, and the setter, `setNumber()`, in order to read or update the variable, for example:

```
SimpleGetterAndSetter obj = new SimpleGetterAndSetter();

obj.setNumber(10);  // OK
int num = obj.getNumber();  // fine
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

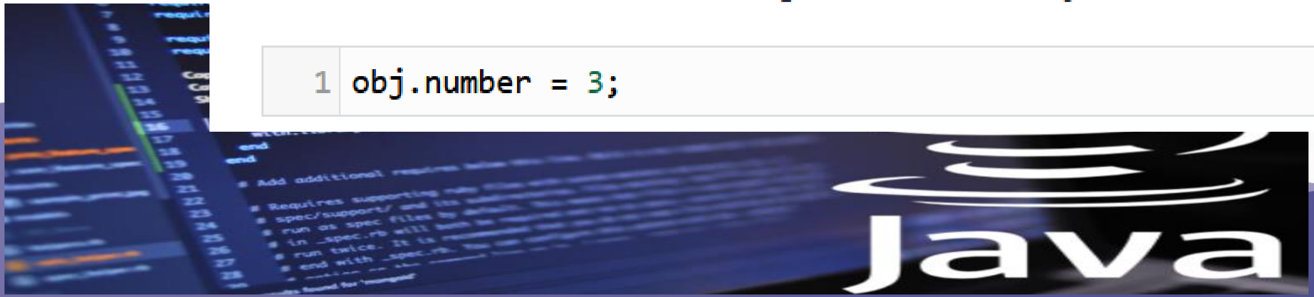
Why do we need Getter & Setter?

By using getter and setter, the programmer can control how their important variables are accessed and updated in the proper manner, such as changing the value of a variable within a specified range. Consider the following code of a setter method:

```
public void setNumber(int num) {
    if (num < 10 || num > 100) {
        throw new IllegalArgumentException();
    }
    this.number = num;
}
```

This ensures that the value of the number is always set between 10 and 100. Suppose the variable number can be updated directly, the caller can set any arbitrary value to it:

```
obj.number = 3;
```

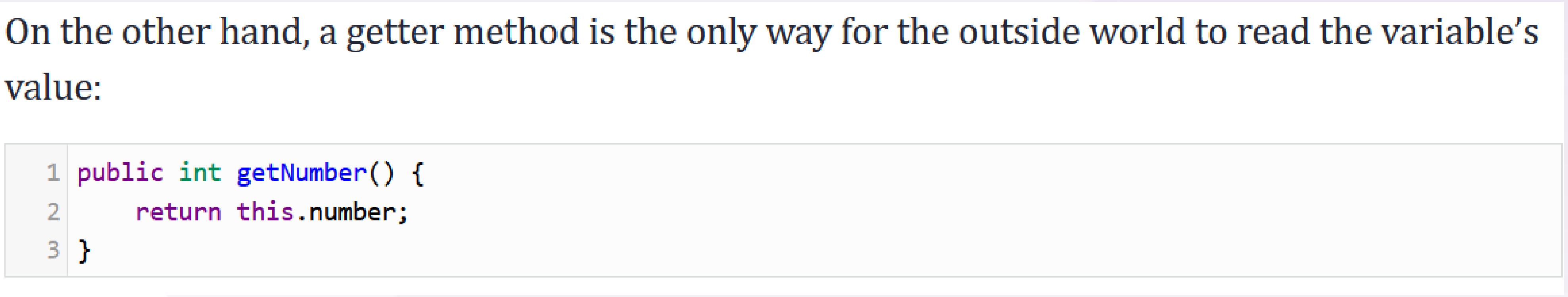


BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Why do we need Getter & Setter?

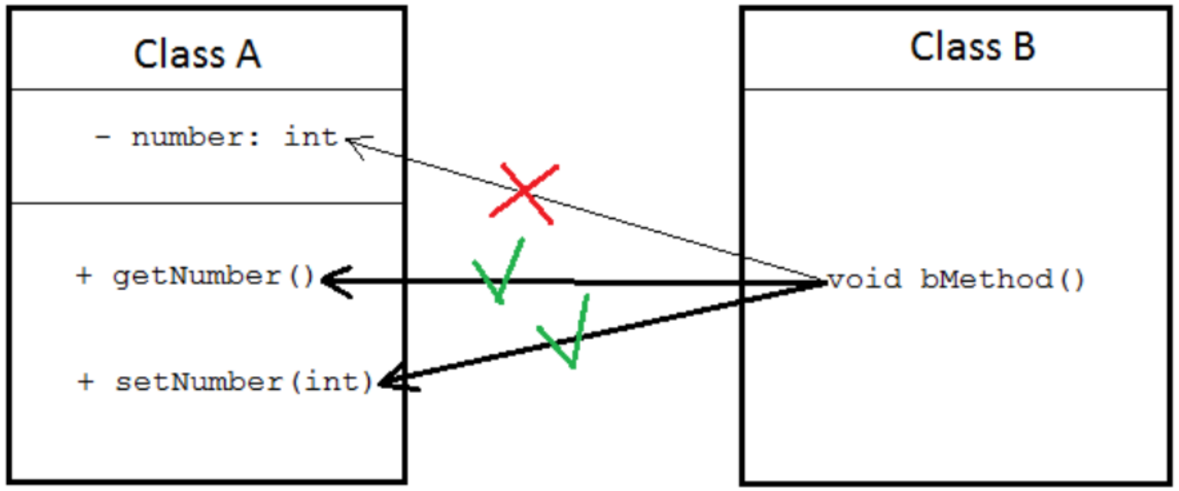

On the other hand, a getter method is the only way for the outside world to read the variable's value:

```java
public int getNumber() {
    return this.number;
}
```

Getters & Setters

Why do we need Getter & Setter?

The following picture illustrates the situation:



So far, the setter and getter methods protect a variable's value from unexpected changes by the outside world — the caller code.

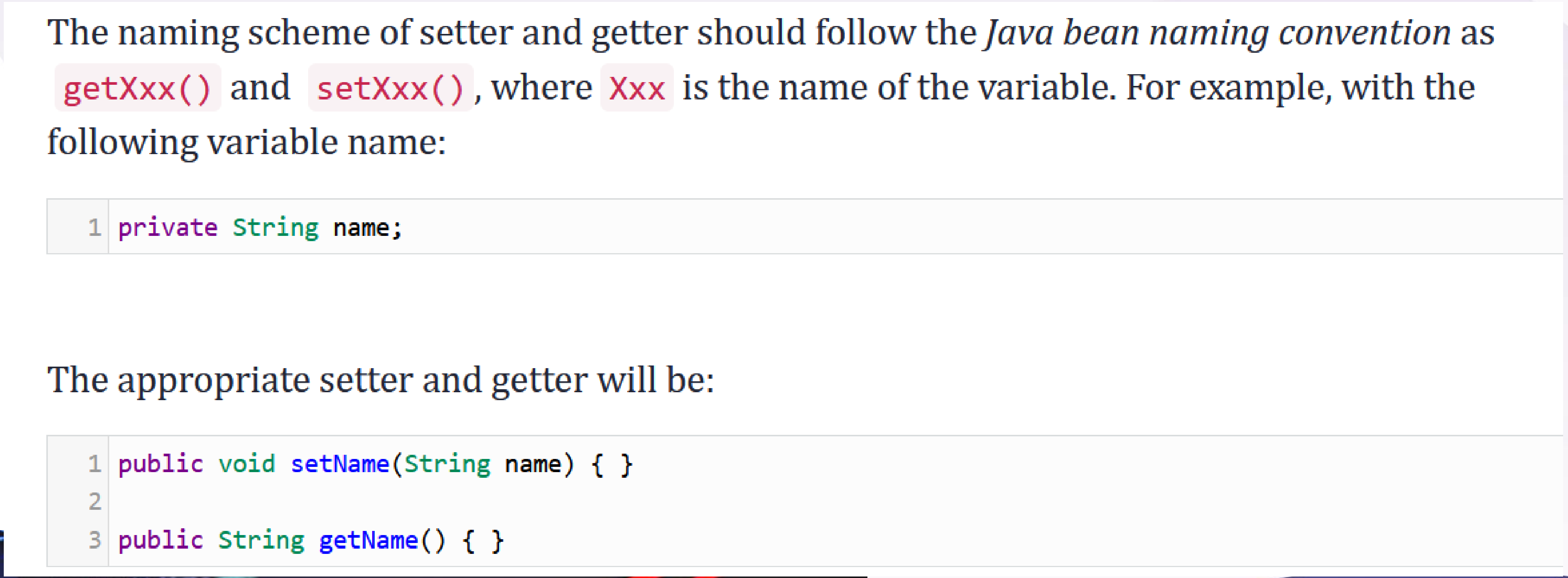
Getters & Setters

Why do we need Getter & Setter?

When a variable is hidden by the private modifier and can be accessed only through getter and setter, it is *encapsulated*. Encapsulation is one of the fundamental principles in object-oriented programming (OOP), thus implementing getter and setter is one of the ways to enforce encapsulation in the program's code.

Getters & Setters

Naming convention for Getter & Setter?

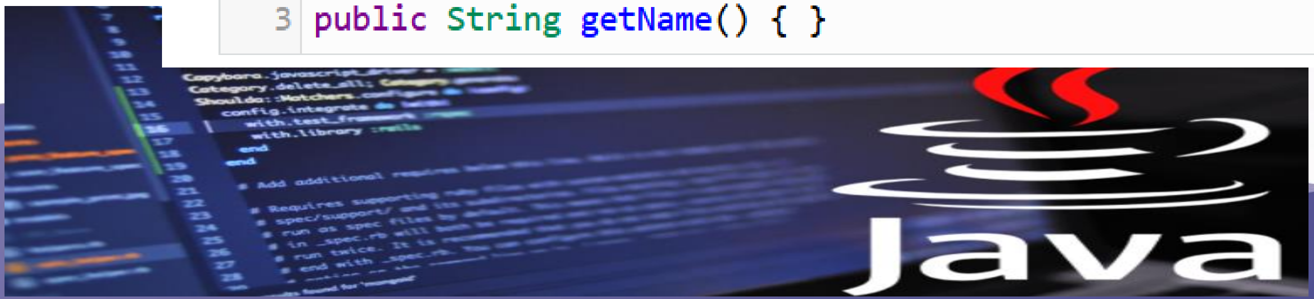


The naming scheme of setter and getter should follow the *Java bean naming convention* as `getXxx()` and `setXxx()`, where `Xxx` is the name of the variable. For example, with the following variable name:

```java
private String name;
```

The appropriate setter and getter will be:

```java
public void setName(String name) { }

public String getName() { }
```



Getters & Setters

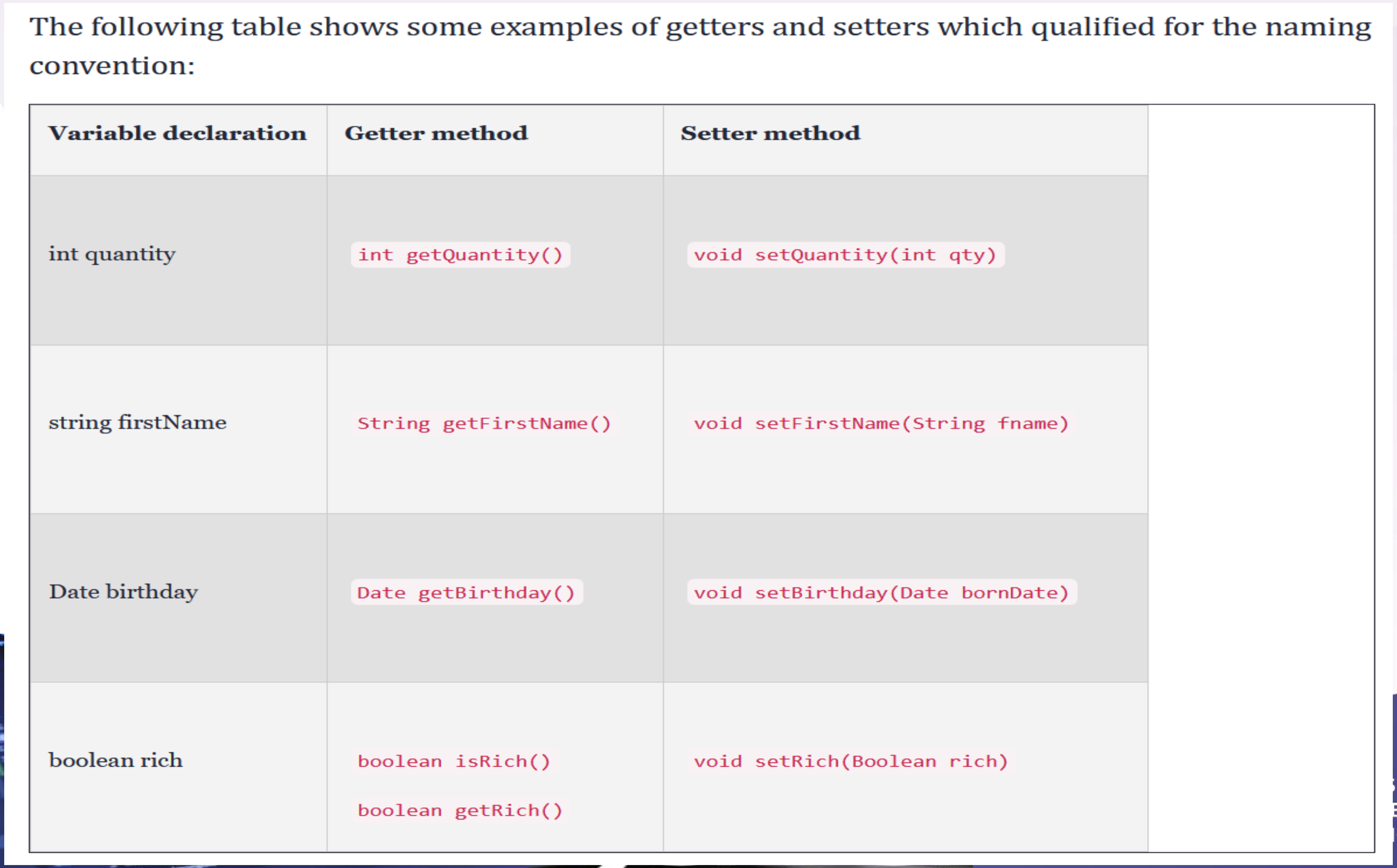
Naming convention for Getter & Setter?

If the variable is of the type boolean, then the getter's name can be either `isXXX()` or `getXXX()`, but the former naming is preferred. For example:

```java
private boolean single;

public String isSingle() { }
```

Getters & Setters

Naming convention for Getter & Setter?



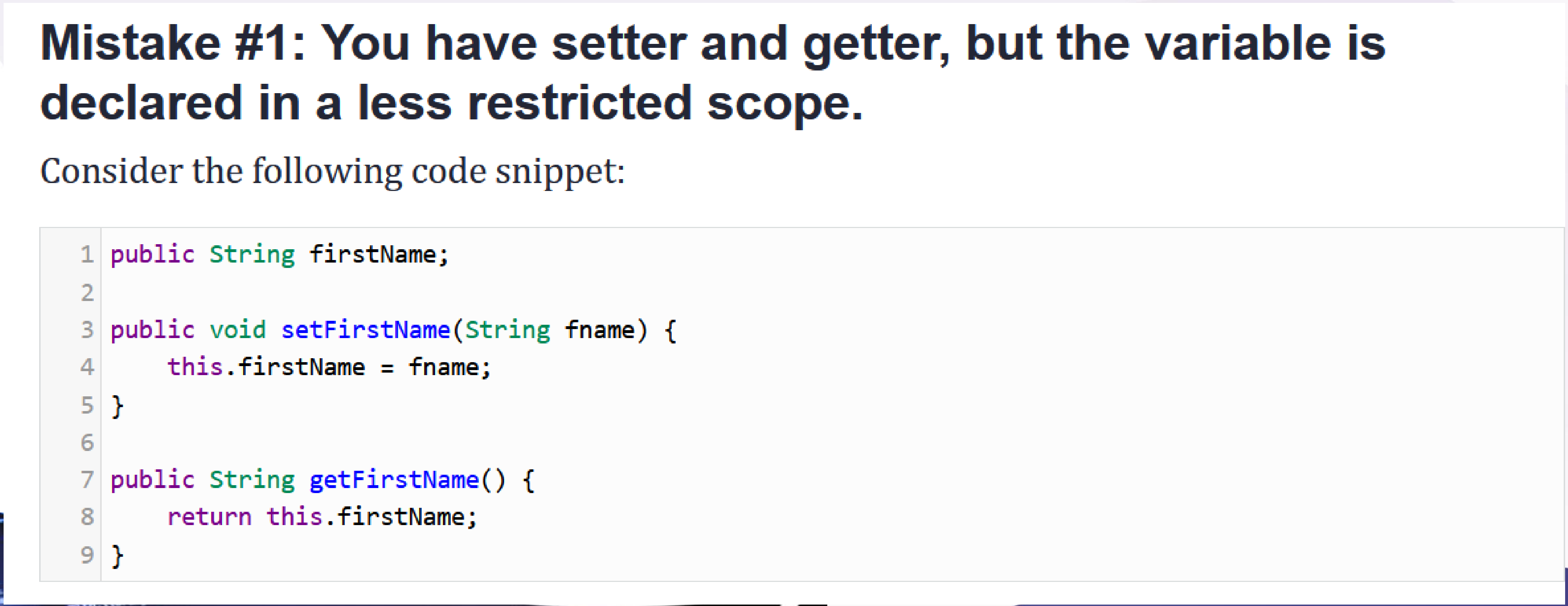
The following table shows some examples of getters and setters which qualified for the naming convention:

Variable declaration	Getter method	Setter method
int quantity	`int getQuantity()`	`void setQuantity(int qty)`
string firstName	`String getFirstName()`	`void setFirstName(String fname)`
Date birthday	`Date getBirthday()`	`void setBirthday(Date bornDate)`

boolean rich	`boolean isRich()`
	`boolean getRich()`
		`void setRich(Boolean rich)`

Getters & Setters

Mistakes using Getter & Setter?



Mistake #1: You have setter and getter, but the variable is declared in a less restricted scope.

Consider the following code snippet:

```java
public String firstName;

public void setFirstName(String fname) {
    this.firstName = fname;
}

public String getFirstName() {
    return this.firstName;
}
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Mistakes using Getter & Setter?

The variable `firstName` is declared as public, so it can be accessed using the dot (.) operator directly, making the setter and getter useless. A workaround for this case is using more restricted access modifier such as protected and private:

```java
private String firstName;
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Mistakes using Getter & Setter?

Mistake #2: Assign object reference directly in the setter

Considering the following setter method:

```java
private int[] scores;

public void setScores(int[] scr) {
    this.scores = scr;
}
```

The following code demonstrates this problem:

```java
int[] myScores = {5, 5, 4, 3, 2, 4};

setScores(myScores);

displayScores();

myScores[1] = 1;

displayScores();
```

Getters & Setters

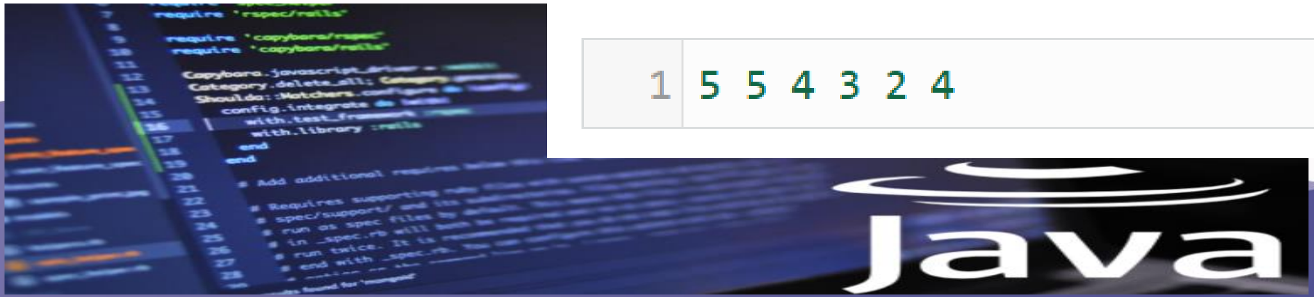
Mistakes using Getter & Setter?

An array of integer numbers, `myScores`, is initialized with 6 values (line 1) and the array is passed to the `setScores()` method (line 2). The method `displayScores()` simply prints out all scores from the array:

```
1 public void displayScores() {
2     for (int i = 0; i < this.scores.length; i++) {
3         System.out.print(this.scores[i] + " ");
4     }
5     System.out.println();
6 }
```

Line 3 will produce the following output:

```
5 5 4 3 2 4
```



Getters & Setters

Mistakes using Getter & Setter?

These are all the elements of the `myScores` array. Now, in line 4, we can modify the value of the 2nd element in the `myScores` array as follows:

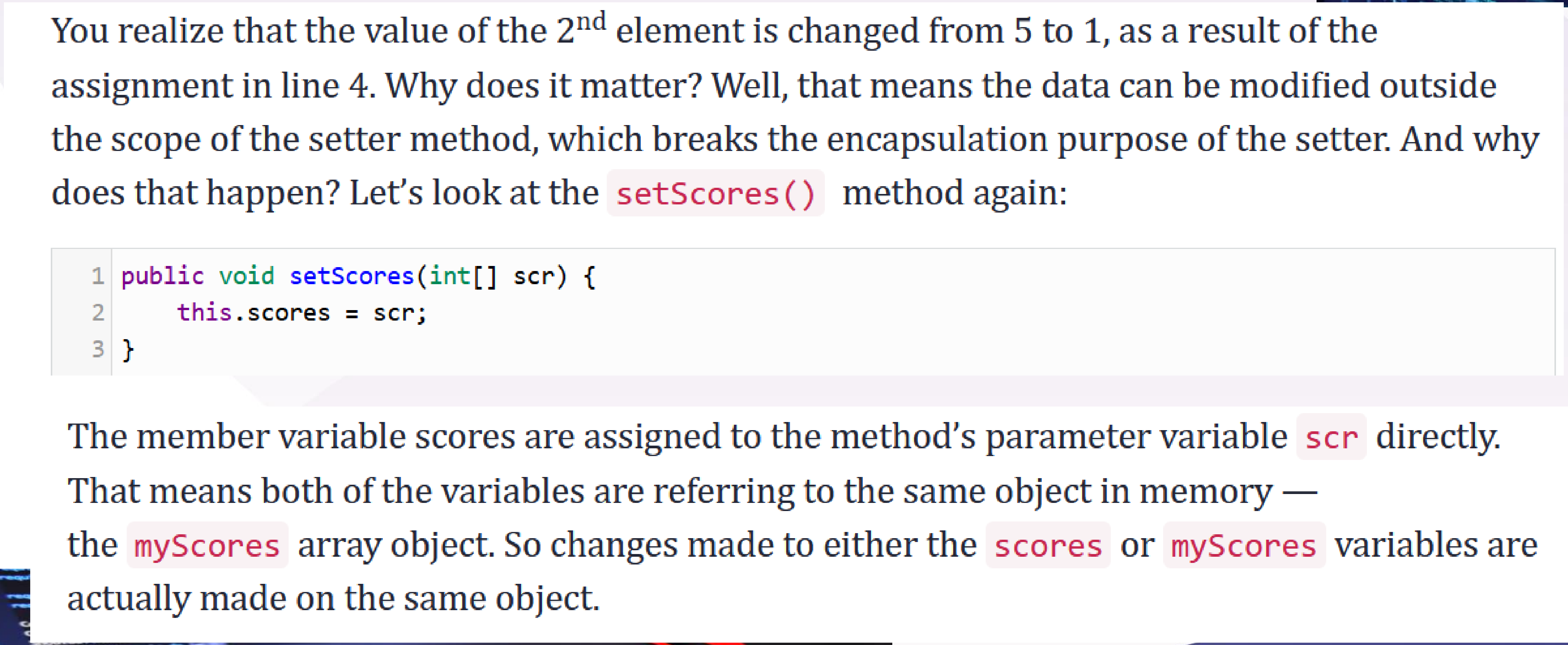
```
myScores[1] = 1;
```

What will happen if we call the method `displayScores()` again at line 5? Well, it will produce the following output:

```
5 1 4 3 2 4
```

Getters & Setters

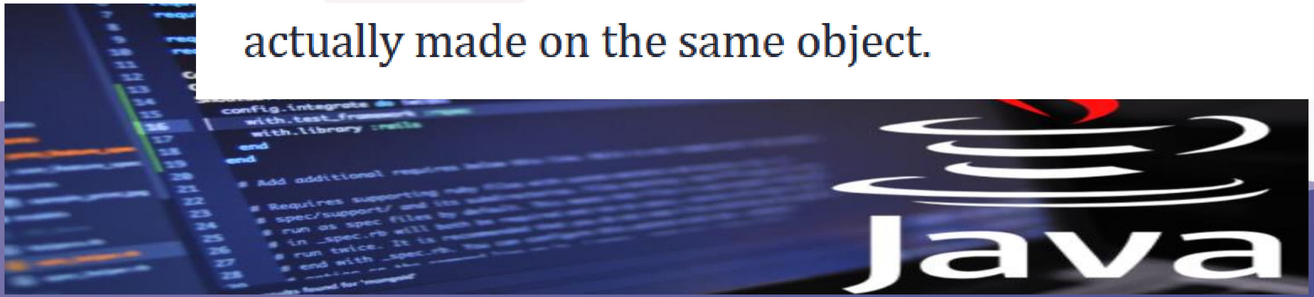
Mistakes using Getter & Setter?



You realize that the value of the 2nd element is changed from 5 to 1, as a result of the assignment in line 4. Why does it matter? Well, that means the data can be modified outside the scope of the setter method, which breaks the encapsulation purpose of the setter. And why does that happen? Let's look at the `setScores()` method again:

```java
public void setScores(int[] scr) {
    this.scores = scr;
}
```

The member variable scores are assigned to the method's parameter variable `scr` directly. That means both of the variables are referring to the same object in memory — the `myScores` array object. So changes made to either the `scores` or `myScores` variables are actually made on the same object.

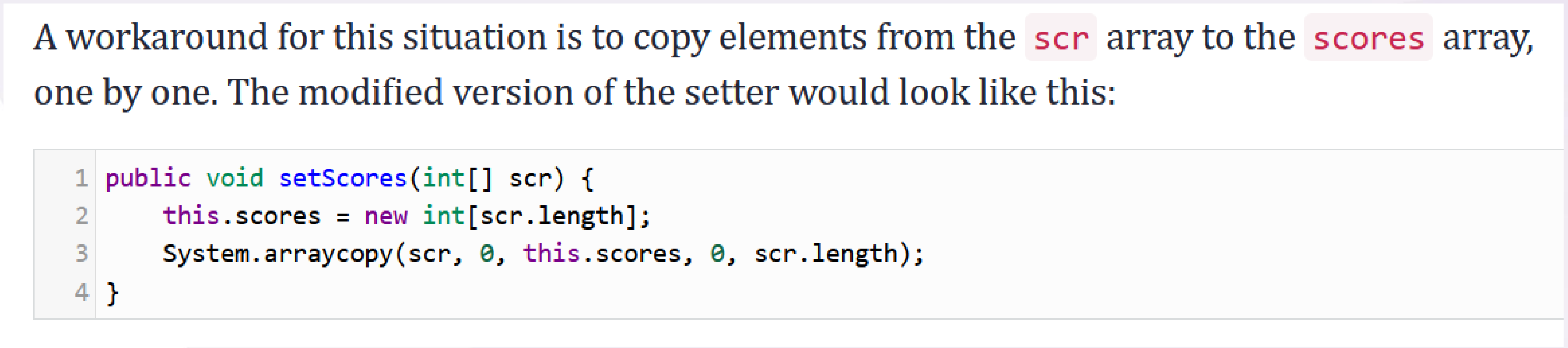


BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

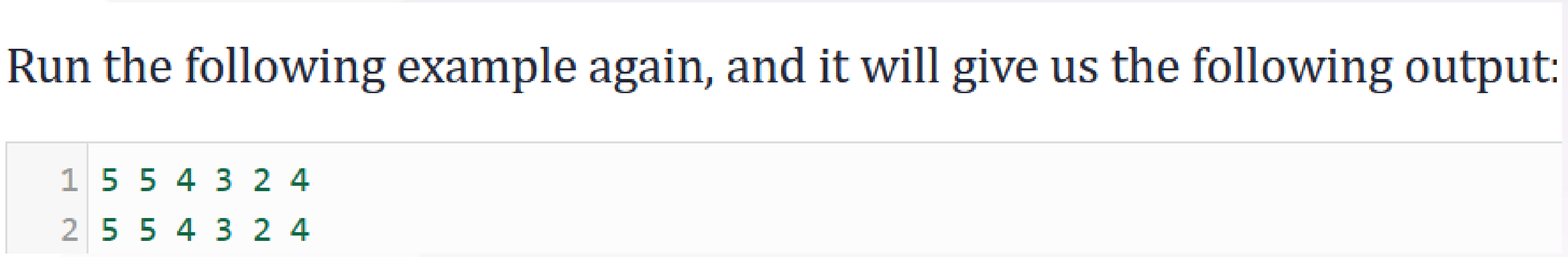
Getters & Setters

Mistakes using Getter & Setter?



A workaround for this situation is to copy elements from the `scr` array to the `scores` array, one by one. The modified version of the setter would look like this:

```java
public void setScores(int[] scr) {
    this.scores = new int[scr.length];
    System.arraycopy(scr, 0, this.scores, 0, scr.length);
}
```



Run the following example again, and it will give us the following output:

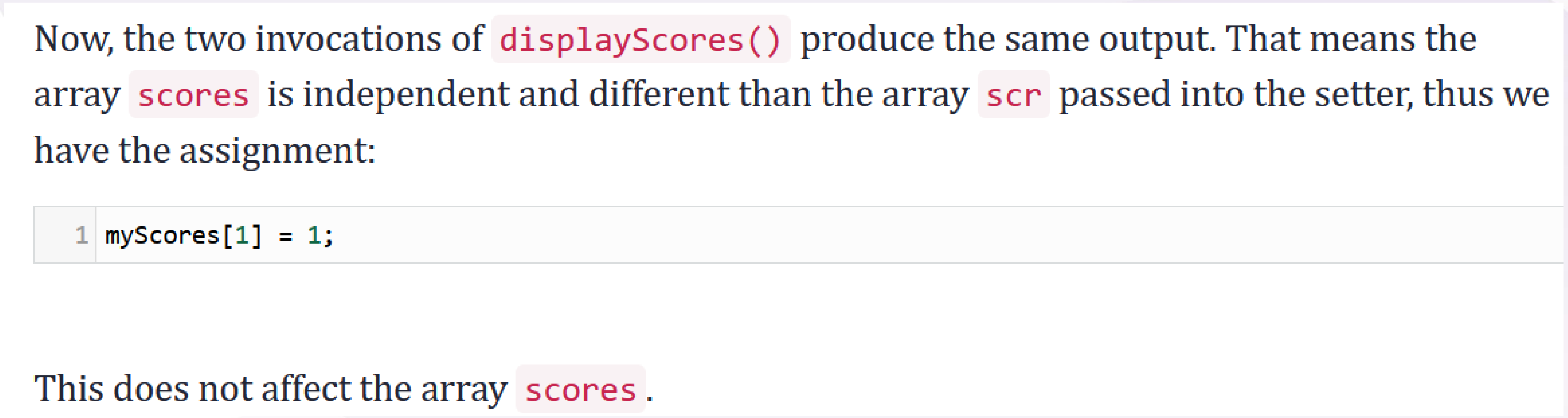
```
5 5 4 3 2 4
5 5 4 3 2 4
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Mistakes using Getter & Setter?



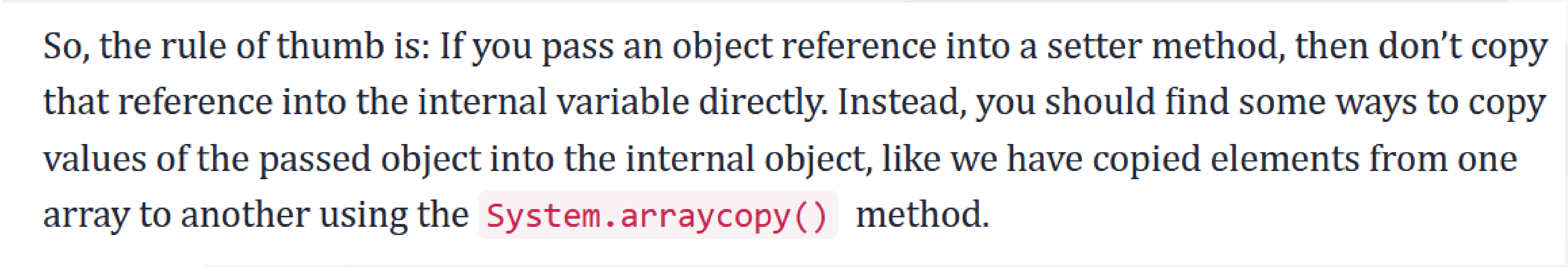
Now, the two invocations of `displayScores()` produce the same output. That means the array `scores` is independent and different than the array `scr` passed into the setter, thus we have the assignment:

```
myScores[1] = 1;
```

This does not affect the array `scores`.

Getters & Setters

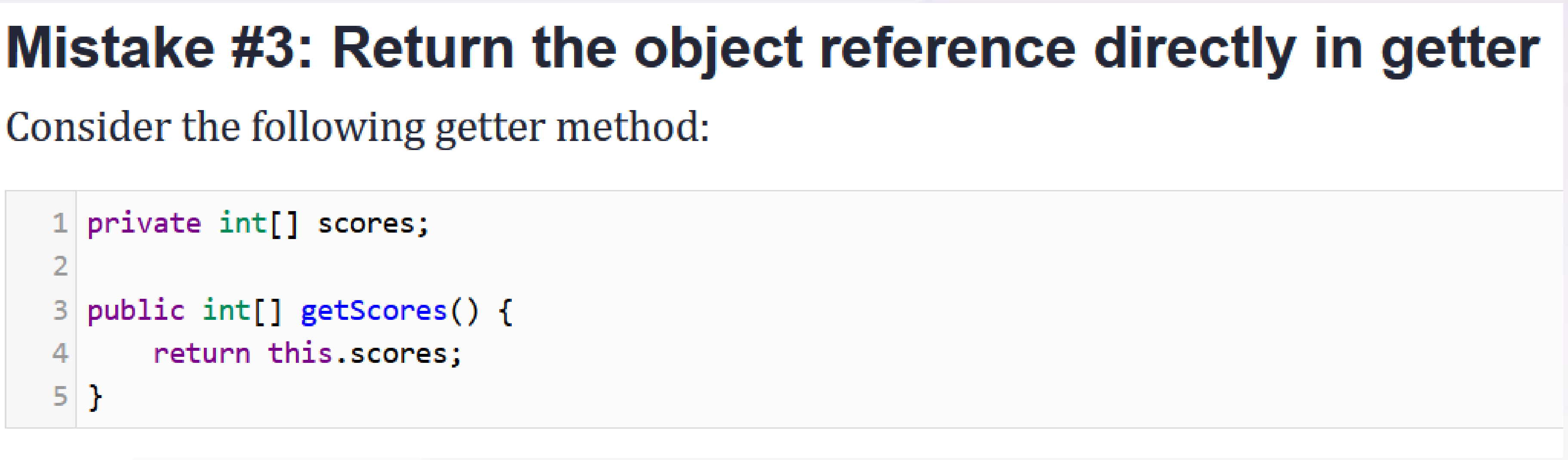
Mistakes using Getter & Setter?



So, the rule of thumb is: If you pass an object reference into a setter method, then don't copy that reference into the internal variable directly. Instead, you should find some ways to copy values of the passed object into the internal object, like we have copied elements from one array to another using the `System.arraycopy()` method.

Getters & Setters

Mistakes using Getter & Setter?



Mistake #3: Return the object reference directly in getter

Consider the following getter method:

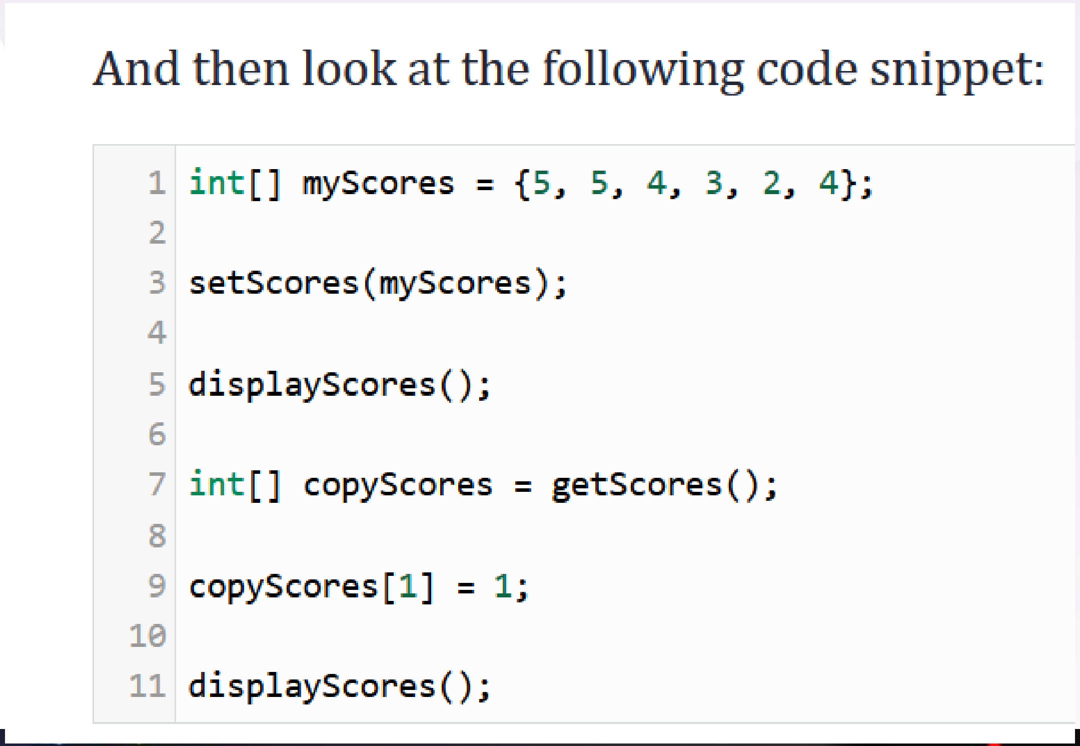
```java
private int[] scores;

public int[] getScores() {
    return this.scores;
}
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

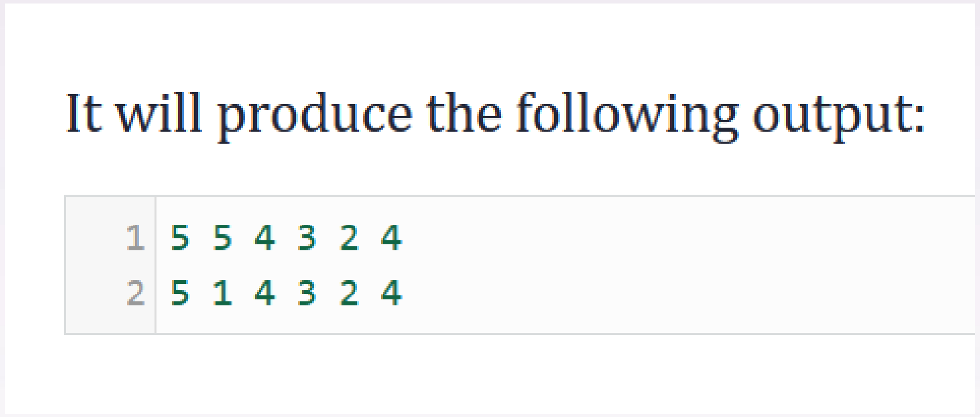
Getters & Setters

Mistakes using Getter & Setter?



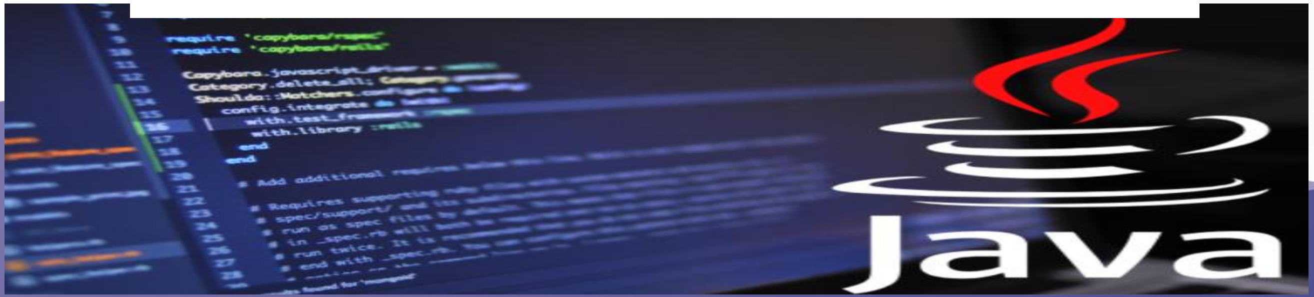
And then look at the following code snippet:

```java
int[] myScores = {5, 5, 4, 3, 2, 4};

setScores(myScores);

displayScores();

int[] copyScores = getScores();

copyScores[1] = 1;

displayScores();
```



It will produce the following output:

```
5 5 4 3 2 4
5 1 4 3 2 4
```



Getters & Setters

Mistakes using Getter & Setter?

As you notice, the 2nd element of the array `scores` is modified outside the setter, in line 5. Because the getter method returns the reference of the internal variable scores directly, the outside code can obtain this reference and make a change to the internal object.

A workaround for this case is that, instead of returning the reference directly in the getter, we should return a copy of the object. This is so that the outside code can obtain only a copy, not the internal object. Therefore, we modify the above getter as follows:

```java
public int[] getScores() {
    int[] copy = new int[this.scores.length];
    System.arraycopy(this.scores, 0, copy, 0, copy.length);
    return copy;
}
```

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Mistakes using Getter & Setter?

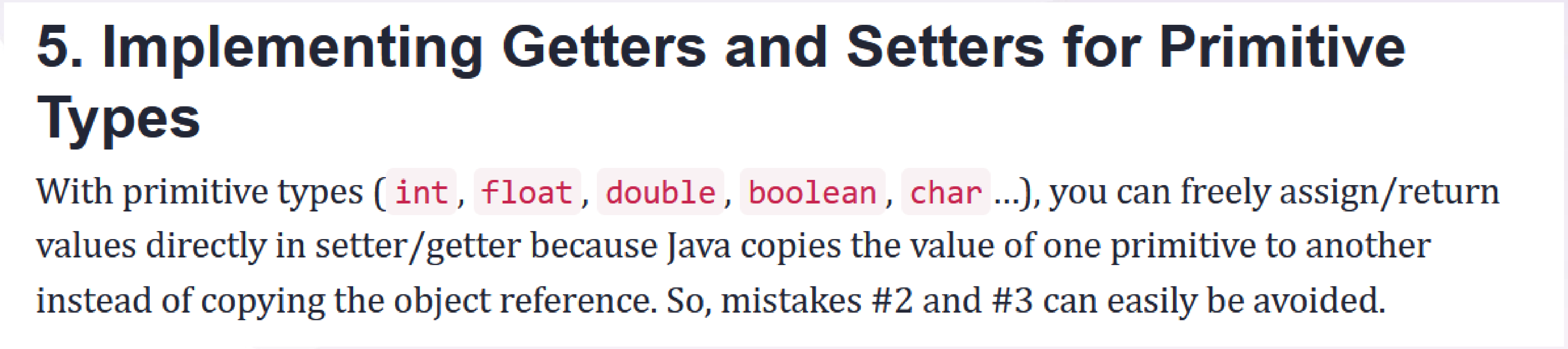
So the rule of thumb is: Do not return a reference of the original object in the getter method. Instead, it should return a copy of the original object.

BASIS INTERNATIONAL™ SCHOOLS · CHINA

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Mistakes using Getter & Setter?



5. Implementing Getters and Setters for Primitive Types

With primitive types (`int`, `float`, `double`, `boolean`, `char` ...), you can freely assign/return values directly in setter/getter because Java copies the value of one primitive to another instead of copying the object reference. So, mistakes #2 and #3 can easily be avoided.

Getters & Setters

Mistakes using Getter & Setter?

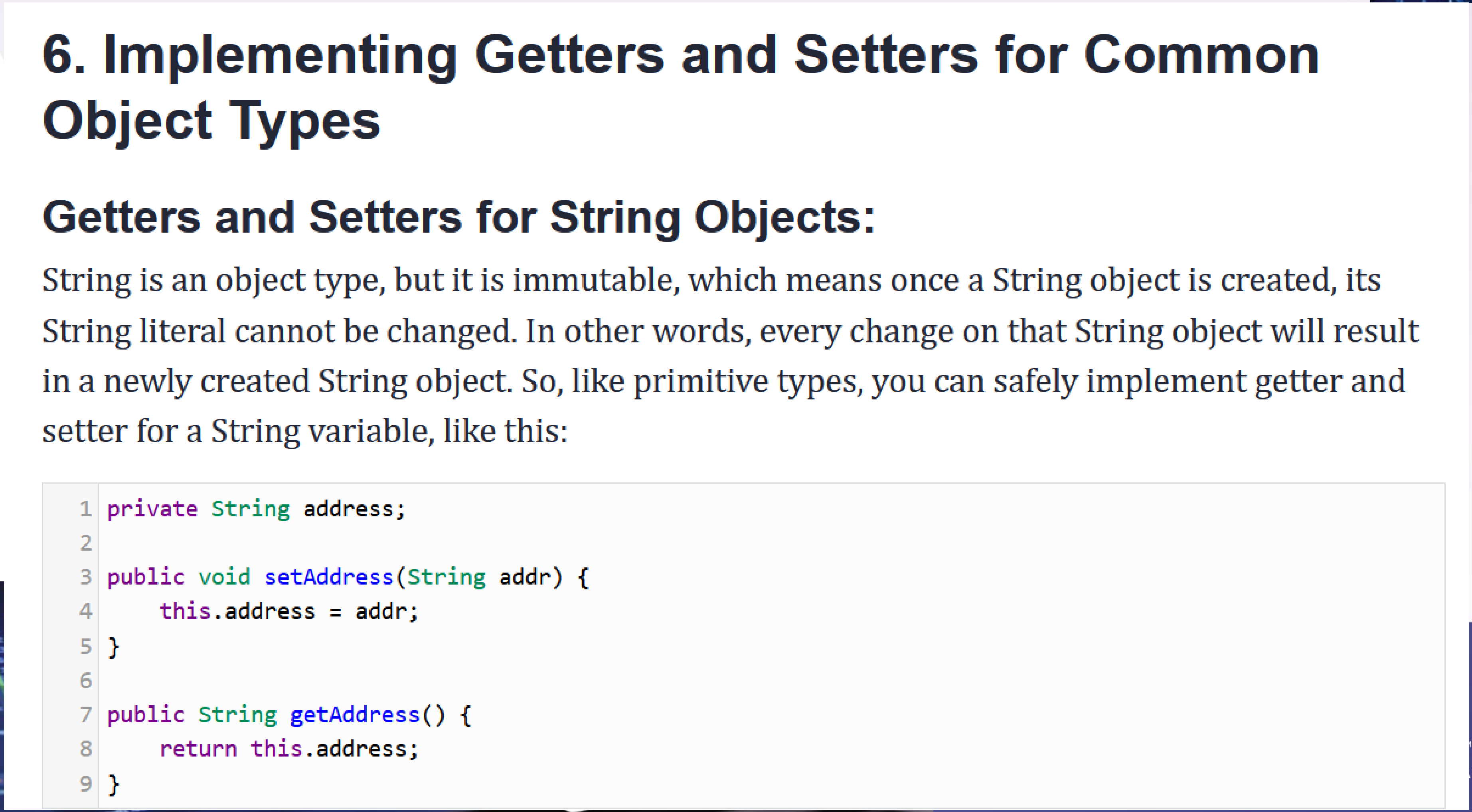
For example, the following code is safe because the setter and getter are involved in a primitive type of `float`:

```java
private float amount;

public void setAmount(float amount) {
    this.amount = amount;
}

public float getAmount() {
    return this.amount;
}
```

So, for primitive types, there is no special trick to correctly implement the getter and setter.

Getters & Setters

Mistakes using Getter & Setter?



6. Implementing Getters and Setters for Common Object Types

Getters and Setters for String Objects:

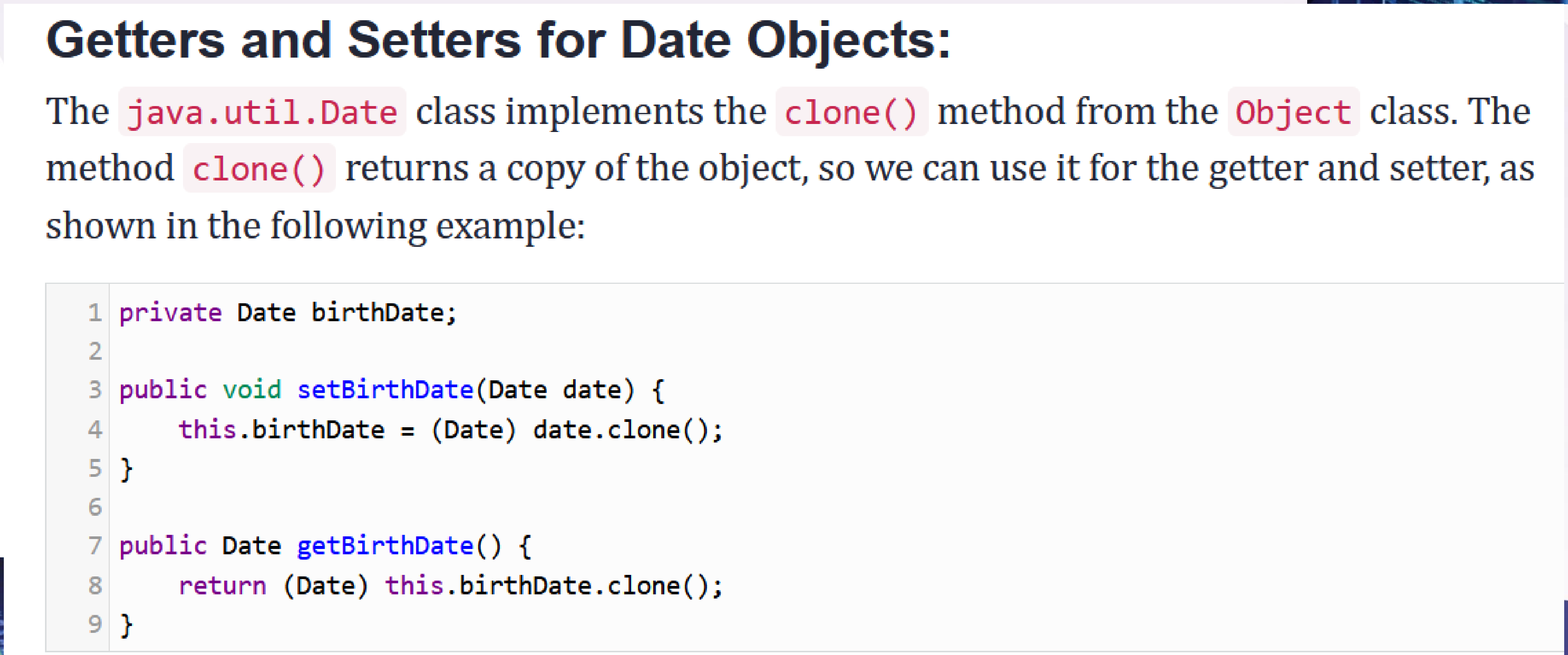
String is an object type, but it is immutable, which means once a String object is created, its String literal cannot be changed. In other words, every change on that String object will result in a newly created String object. So, like primitive types, you can safely implement getter and setter for a String variable, like this:

```java
private String address;

public void setAddress(String addr) {
    this.address = addr;
}

public String getAddress() {
    return this.address;
}
```

BASIS BILINGUAL™ SCHOOLS · CHINA

Getters & Setters

Mistakes using Getter & Setter?



Getters and Setters for Date Objects:

The `java.util.Date` class implements the `clone()` method from the `Object` class. The method `clone()` returns a copy of the object, so we can use it for the getter and setter, as shown in the following example:

```java
private Date birthDate;

public void setBirthDate(Date date) {
    this.birthDate = (Date) date.clone();
}

public Date getBirthDate() {
    return (Date) this.birthDate.clone();
}
```