7.15-Getters & Setters

In Java, **Getters** and **Setters** are special methods used to access and modify private variables within a class. Since private variables are only accessible within the class they are declared, outside classes cannot directly access them. However, by providing public getter and setter methods, you can control how these variables are accessed and modified.

Why Use Getters and Setters?

- **Encapsulation**: Getters and setters allow you to encapsulate the data within your class. This means that the internal representation of an object is hidden from the outside world, and access to this data is controlled.
- **Data Protection**: By using getters and setters, you can ensure that the data is only accessed or modified in a controlled manner, preventing unauthorized access or changes.
- **Code Maintainability**: If the internal structure of the class changes, you can modify the getter and setter methods without changing the code that uses the class.

Syntax

Both getter and setter methods start with either get or set, followed by the name of the variable with the first letter capitalized. The getter method returns the value of the variable, while the setter method assigns a new value to the variable.

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

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

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

How Getters and Setters Work

- **Getter in Java**: A getter method, also known as an accessor, is used to retrieve the value of a private variable. The method returns the value of the variable, which could be of any data type like int, String, double, float, etc. The method name conventionally starts with "get" followed by the capitalized variable name.
- **Setter in Java**: A setter method, also known as a mutator, is used to update the value of a private variable. The method sets the value of the variable and conventionally starts with "set" followed by the capitalized variable name.

Example

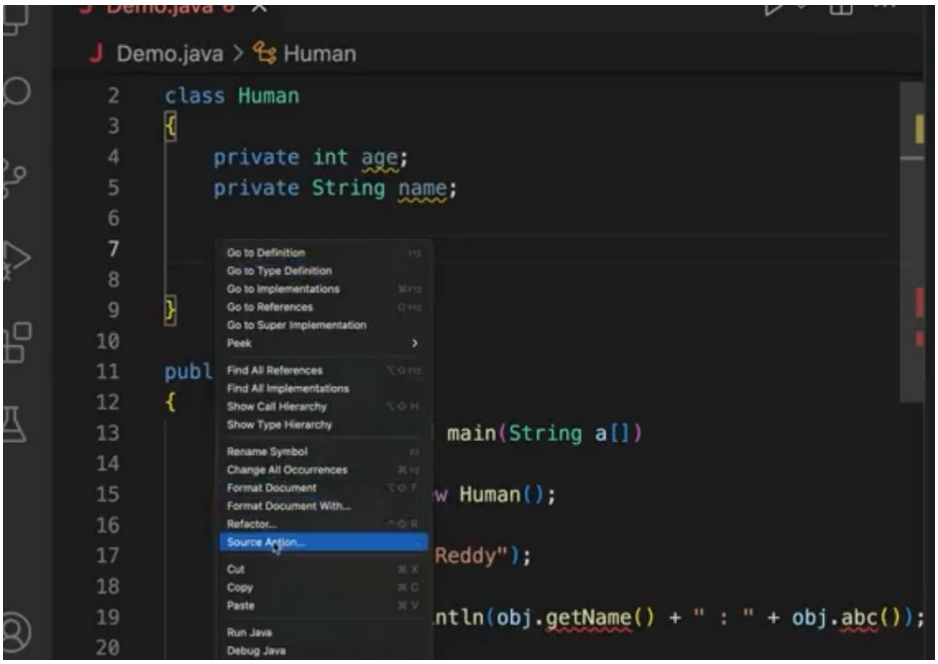
Here's an example using a Vehicle class to demonstrate getters and setters:

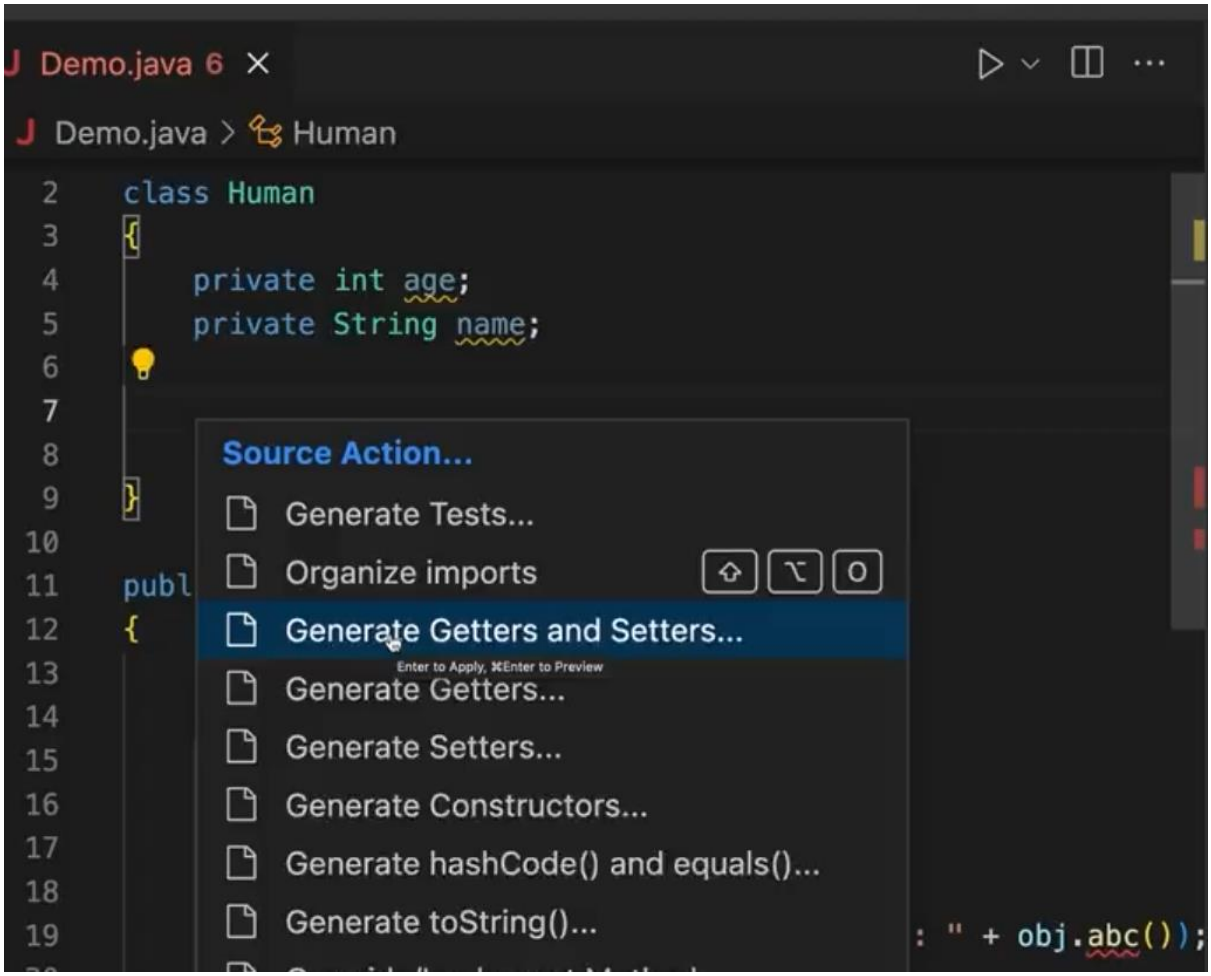
```
public class Vehicle {
    private String color;

    // Getter
    public String getColor() {
        return color;
    }

    // Setter
    public void setColor(String c) {
        this.color = c;
    }
}
```

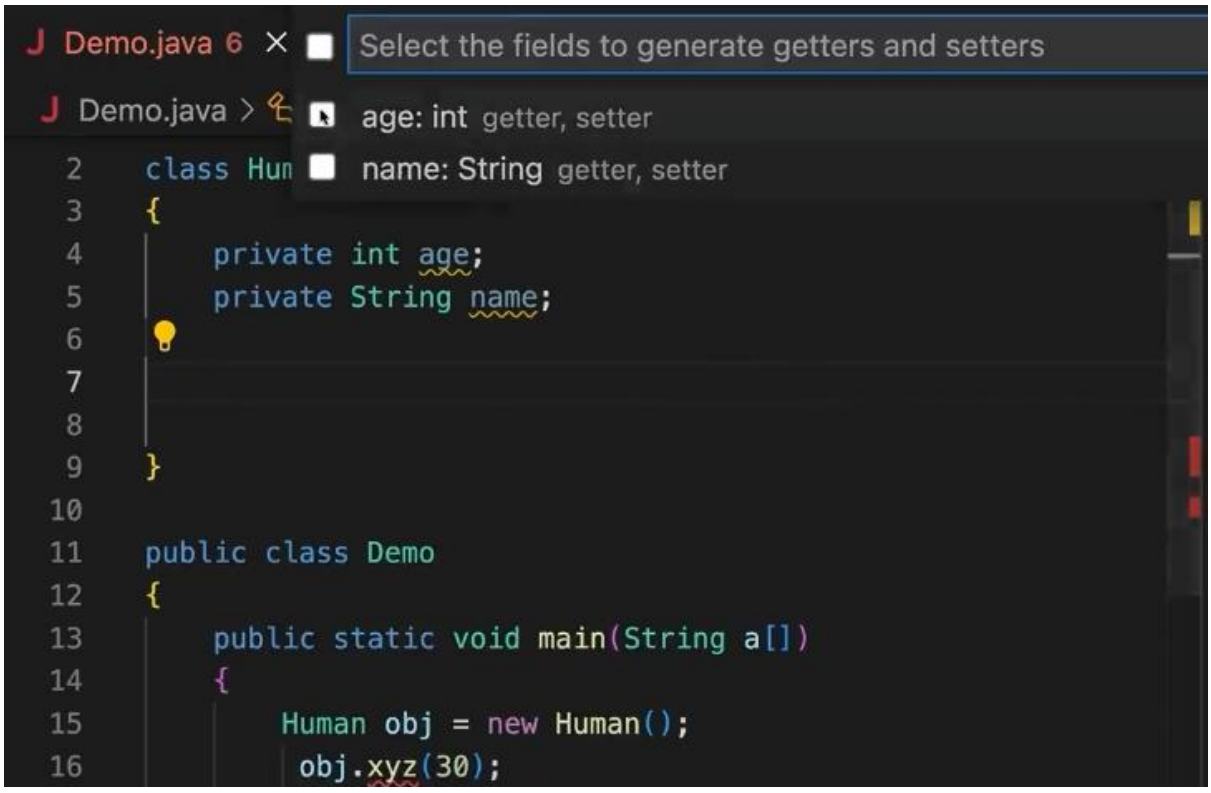
In this example:

- The getColor method returns the value of the color variable.
- The setColor method sets the value of the color variable.

Using Getters and Setters in IDEs




Demo.java 6
Demo.java > Human
class Human
{
private int age;
private String name;
}
publ
{
Source Action...
Generate Tests...
Organize imports
Generate Getters and Setters...
Enter to Apply, ⌘Enter to Preview
Generate Getters...
Generate Setters...
Generate Constructors...
Generate hashCode() and equals()...
Generate toString()...
: " + obj.abc());


Demo.java 6
Select the fields to generate getters and setters
Demo.java
age: int getter, setter
name: String getter, setter
class Hun
{
private int age;
private String name;
}
public class Demo
{
public static void main(String a[])
{
Human obj = new Human();
obj.xyz(30);

```java
class Human
{
    private int age;
    private String name;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

Most modern Integrated Development Environments (IDEs) provide built-in support for generating getters and setters. This can save you time and ensure that the methods follow best practices. You can select the variables you want to generate methods for, and the IDE will create the code for you.

Important Considerations

- **Not Mandatory**: While it's a common practice to name these methods as get and set, it's not a strict requirement. You can name them anything, but following conventions makes the code more readable and understandable to other developers.
- **Flexibility**: Getters and setters offer flexibility in controlling how the variables are accessed and modified. For example, you can add validation logic in a setter to ensure that only valid data is assigned to a variable.