Welcome to the first lab of the lesson! In this lab, you will change the code in the **`SideKick`** *class* to use encapsulation by leveraging the private access specifier. This will help protect its data and methods, allowing you to hide or show specific properties and methods as needed.

Toni has asked you to help code his robot, *SideKick*. You have already coded the recharging of *Sidekick's* battery, which is included in the starter code. Now, Toni has requested that you provide a menu system for cleaning and cooking. *SideKick* needs to do something when its battery is fully charged!

Goal

Enhance *SideKick's* functionality by implementing encapsulation and data hiding. You will protect the properties and methods of the **`SideKick`** class, ensure secure and validated operations and values, and expand its capabilities to include both cleaning and cooking tasks. By the end of this lab, you'll have a deeper understanding of encapsulation principles and how to apply them to create robust, maintainable code.

You'll enhance *SideKick* using encapsulation and data hiding. Here's the plan:

- Implement encapsulation:
    - Modify *SideKick.java* to use the private access specifier for properties and methods.
    - Create **`getter`** methods to provide controlled access to private properties.
    - Test encapsulation by accessing properties via **`getter`** methods.
- Secure and validate operations
    - Create **`setter`** methods to validate and secure property values.
    - Use the **`setter`** method to modify **`modeOfOperation`** and ensure it accepts valid values. If an incorrect value is sent then it will set it to a correct default value.
- Expand functionality
    - Implement the **`cookFood()`** method in *SideKick.java* to add cooking functionality.
    - Protect the new method with the private access specifier.
    - Use the **`takeAction()`** method to execute tasks based on the value of the property **`modeOfOperation`**, accessing the value of **`modeOfOperation`** using the getter method.

By following these steps, you'll enhance *SideKick's* capabilities while maintaining secure and robust code through the correct use of encapsulation.

🖥️ Note: When encountering this icon, it's time to get into your IDE and start coding!

In your lab environment, open IntelliJ by double-clicking on the icon.

The `main` method of the `main` class is placed in a file named *Main.java*.

Tip

Always follow best practice when naming files: The name of the class should be the same as the file name.

As soon as you open Intellij, you will be presented with two files:

- ***SideKick.java***
- ***Main.java***

The *Main.java* file will have its own TODOs. You will be given instructions as to whether to move to the *SideKick.java* file or the *Main.java* file. The term TODO lets you know your task and the position within the program where you will

type it in. The term ALL other TODO's lets you know that there will be different TODOs in the starter code.

If you do not want unwanted changes to the properties of a class or want to prevent access to a property or method from outside the definition of the class, you use the keyword private. For more information, refer to the video *Encapsulation and data hiding in action*.

Now that you're all set up, it's time to dig in and help Toni code *SideKick* to perform some tasks! Specifically, Toni needs your help to manage access to *SideKick's* properties. Let's get coding!

🖥️ It's time to get coding!

- **🖥️ TODO 1: Go to the *SideKick.java* file and observe the code to see how the property `modeOfOperation` is already protected from outside access.**

```
/** TODO 1: Make property modeOfOperation protected from outside access **/

/**

 * Property to check the mode - default is cleaning

 * +-------------+---------------+

 * | Mode Value  | Action        |

 * +-------------+---------------+

 * |     1       |  Cleaning     |

 * |     2       |  Cooking      |

 * |     3       |  Re-charge    |

 * |  Others     | Not Supported |

 * +-------------+---------------+

 **/

private int modeOfOperation;
```

- 🖥️ TODO 2: Go to the *Main.java* file. Create an object of the class `SideKick` using the name `sideKickObject`. For more information, refer to the video *Classes and Objects*.
- 🖥️ TODO 3: In the *Main.java* file, try to access the property `modeOfOperation` using the `sideKickObject`, from outside the `SideKick` class. You will receive an error message from IntelliJ. When you run the main program, you will receive the error message: “modeOfOperation has private access”. Performing this task helps you to understand that when a property is protected by using private, which you did in TODO 1, it cannot be accessed from outside the curly braces of the class where it is made private, even through objects of the class. The error confirms that the property is protected, and we now need a proper way to access it.
- 🖥️ TODO 4: Let us comment out the reason for the error for now. In *Main.java*, comment out the line created by TODO 3 and run the program again. The program runs without errors, but you will observe that nothing is displayed.

So, why is there no display? When you comment out the line that tries to access `modeOfOperation` directly, the program no longer has code to display anything. This is because you commented out the display to resolve the error of trying to access a private property.

The convention states that we should create a getter method for accessing the values and a `setter` method to change the values. If confused, refer to the *Encapsulation and data hiding in action* lab.

Important

A `getter` and `setter` method should always be public to allow it to be accessed from outside the class. Do not make it private. If you do, you will not be able to access it from outside the class in the main method.

```
public class Main {

    public static void main(String[] args) {



        /** TODO 2: Create an object of the SideKick class named "sideKickObject"
**/

        /** TODO 3: Try accessing the property "modeOfOperation" using the dot
operator
```

```
        *          like:

        *          System.out.println("Mode of operation is: " +
sideKickObject.modeOfOperation);

        *

        **/
```

- 🖥️ TODO 5: Now, since the property `modeOfOperation` of the class `SideKick` is private, you must provide a way to allow the user to get its value from outside the class. How do you do this?

You can do this using `getter` methods. For more information, refer to the video *Encapsulation and data hiding in action*.

So why don't you go ahead and create a `getter` method which returns the current value of `modeOfOperation`. To do this, go to the file *SideKick.java* and implement TODO 4.

```
 // accessor/getter method  for modeOfOperation

/** TODO 5: Implement a getter for "modeOfOperation" using Java convention   **/
```

- 🖥️ TODO 6: Go to *Main.java*. Now that you have a `getter` method, why don't you try using that to access the property of the class using the `getter` method?

```
  *         "sideKickObject" within a println() using the getter method of the
object

  *         you created in TODO 5

  **/

  /** TODO 6: Display the current value of the property "modeOfOperation"  of the
object
```

Run the main program. You will now see the value of the private property.

Congratulations! You have successfully protected access to a property named `modeOfOperation` in the class `SideKick` in TODO 1. In TODO 3, you understood that such a property becomes inaccessible from outside the class. In TODO 5, you created a method to get the value or a `getter`. Now, in TODO 6, you used that `getter` from outside the class to access the private property indirectly, using the public `getter` method, via objects of the class.

Alright, so now you've made `modeOfOperation` private to protect it, but how will you allow Toni to access it from outside the class? That's right; you need to use a `getter` method. After creating the `getter` method, you have access to `modeOfOperation` and can proceed with implementing the menu system.

That is excellent work so far, now let's keep going!
Toni needs your help to ensure that *SideKick*'s properties are secure and validated. By following these steps, you'll set up a menu system for cleaning and cooking, validate property values, and understand the importance of access specifiers. Implementing validation is crucial for maintaining data integrity, enhancing security, improving user experience, and ensuring the robustness of the SideKick system. Proper validation prevents errors from invalid data, reduces security risks, helps users understand and correct mistakes, and makes the code easier to maintain and debug.

🖥️ It's time to get coding!

- 🖥️ TODO 7: Go to the *SideKick.java* file and create the setter method in the same way as you did for the getter method. Ensure that you follow conventions and that the setter method is public. For more information on setter methods, refer to the video ***Encapsulation and data hiding in action***.
- 🖥️ TODO 8: In the *SideKick.java* file, check the setter method you just created in TODO 7. If the value of the parameter being passed is not between 1 and 3, set it to 1.

Why did you do this? Well, one important reason to create `setter` methods and not allow direct access is to ensure that the property's value is valid. Since the property's value can only be between 1 and 3, if an invalid value such as 200 is sent, you are preventing the value from representing an invalid value and setting it to a default value of 1.

TIP

A `setter` should always ensure that the value being assigned to a property is a valid value for the property, with reference to the context of the class or program.

```
    // mutator/setter method for modeOfOperation
    /** TODO 7: Implement a setter for "modeOfOperation" using Java convention
**/



    /** TODO 8: In the setter method you created in TODO 6 put in a condition

     *          to ensure that if the value being passed is not between 1 and 3

     *          then set it to the default of 1 - which represents cleaning

     **/
```

- 🖥️ **TODO 9: Go to *Main.java*. Set the value of the property `modeOfOperation` to have a value of 3, using the `setter` method you created in TODO 7 and TODO 8.**
- 🖥️ **TODO 10: In *Main.java*, use the method `takeAction()` of the object `sideKickObject`, which you created in TODO 2.**

```
    /** TODO 9: Set the value of the property "modeOfOperation" of the object

     *        "sideKickObject", using the setter you created in TODO 8, to a

     *        a value of 3

     **/



    /** TODO 10: Call the method takeAction() of the object "sideKickObject"
**/
```

What did you observe? It executed `case 3` and recharged the battery, right? Why? This is because you changed the value of `modeOfOperation` to 3 in TODO 9, using the public `setter` method.

```
     // method to take action

     public void takeAction() {

        switch (modeOfOperation) {
```

```
        case 1:

            cleanHouse();

            System.out.println("SideKick cleaning completed.");

            break;

        case 2:

            System.out.println("SideKick cooking completed.");

            break;

        case 3:

            rechargeBattery();

            System.out.println("SideKick recharged battery.");

            break;

        default:

            System.out.println("SideKick does not support the operation.");

    }

}
```

- 🖥️ **TODO 11: Return to *SideKick.java* and protect the method `cleanHouse()` from external access. How do you protect properties and methods from external access (such as from access from outside the class)? You can do this by using private in place of public.**

```
/** TODO 11: The method cleanHouse() will only be used internally.

    *           Protect it with "private"

    **/

    // method to clean

  public void cleanHouse() {
        System.out.println("Get the vacuum cleaner.....");

        System.out.println("Put the dust bag in vacuum.....");

        System.out.println("Go to Living room and clean.....");

        System.out.println("Go to bedroom and clean.....");

        System.out.println("Go to kitchen and clean.....");

        System.out.println("Go to bathroom and clean.....");

        System.out.println("Retrieve dust bag from vacuum cleaner and put in
bin.....");

        System.out.println("Go back to Toni.....");

    }
```

- 🖥️ **TODO 12: Go to *Main.java* and use the `setter` method of the property `modeOfOperation` to pass the value 1 and set the value of `modeOfOperation` to 1. Then, call the method `takeAction()`.**

```
/** TODO 12: Set the value of the property "modeOfOperation" of the object

 *          "sideKickObject to a value of 1

 *           using the setter method you created in TODO 7 and TODO 8.

 *           After that is done, call the method takeAction() using object

 *           "sideKickObject" and the dot operator.

 **/
```

Now, run the main program.

Does it make a difference to the execution of the cleaning by *SideKick*? No, it does not. Remember the execution of a method remains the same, irrespective of the access specifier. It just changes the area where the property or method

can be accessed. When you make it private it can be accessed only within the class, but it executes in the same way.

Great work! You've successfully ensured that *SideKick's* properties are validated and secured by using getter and setter methods in combination with the private access specifier. This not only enhances the functionality of the robot but also maintains the integrity of its operations. Keep up the excellent coding!

Now that you've secured and validated *SideKick's* operations, it's time to enhance its functionality even further. Toni needs your help implementing a cooking feature for *SideKick*. Let's make sure *SideKick* can handle both cleaning and cooking tasks!

🖥️ It's time to get coding!

- 🖥️ TODO 13: In *SideKick.java*, implement the cooking method to enable *SideKick* to display the different tasks performed during cooking. Protect the method from access from outside the class. Name this method **`cookFood()`**.

```
// method to cook

/** TODO 13: Create a method to display the tasks SideKick does

 *           in order to cook, like

 *           1. Move to the kitchen

 *           2. Get the vegetables

 *           3. Cut the vegetables

 *           4. Turn on the gas

 *           5. Get the cooking pan and oil ready

 *           6. Cook the food

 *           7. Turn off the gas

 *           8. Get it ready on the plate

 *           9. Go back to Toni

 *

 **/
```

```
}
```

🖥️ TODO 14: In *SideKick.java* call the method **`cookFood()`**, which you created in TODO 13.

Now, how will you integrate this new feature into the main program and ensure that *SideKick* can switch between cleaning and cooking tasks based on the value of the property **`modeOfOperation`**? Let's test this out!

```
  public void takeAction() {

        switch (modeOfOperation) {

            case 1:

                cleanHouse();

                System.out.println("SideKick cleaning completed.");

                break;

            case 2:

                /** TODO 14: call the method cookFood() you created in TODO 13  **/

                System.out.println("SideKick cooking completed.");

                break;

            case 3:

                rechargeBattery();

                System.out.println("SideKick recharged battery.");

                break;

            default:

                System.out.println("SideKick does not support the operation.");

        }

    }
```

- 🖥️ TODO 15: Go to *Main.java*, use the **`setter`** method of the property **`modeOfOperation`** to pass the value 2 and set the value of **`modeOfOperation`** to 2. Then, call the method **`takeAction()`**. Now, run the main program.

```
        /** TODO 15: Set the value of the property "modeOfOperation" of the object
         *          "sideKickObject to a value of 2
         *           using the setter method you created in TODO 7 and TODO 8.
         *           After that is done, call the method takeAction() using object
         *           "sideKickObject" and the dot operator.
         **/
    }
}
```

Fantastic! You've successfully implemented and tested the cooking functionality for *SideKick*.

By protecting methods and properties and adding validation to the setter methods to prevent incorrect data from being assigned to a property, you've ensured that *SideKick's* operations are secure and robust.

In this lab, you successfully helped Toni program *SideKick* by implementing encapsulation through the use of the private access specifier. You learned how to protect the properties and methods of *SideKick* to prevent unwanted access and modification. By creating and using public getter and setter methods, you enabled controlled access to private properties, ensuring their values remain valid. On top of this, you enhanced *SideKick's* capabilities by adding a cooking option.

This lab provided a practical understanding of encapsulation, a key concept in object-oriented programming. You saw how access specifiers like private control access without affecting functionality. This knowledge will help you develop more complex programs and write robust, maintainable code in future projects.