**Encaulation Assignments**

1. **what is encapsulation in java ? why is it called data hiding?**

**Ans**: Encapsulation in Java is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data (attributes) and the methods (functions) that operate on that data within a single unit, known as a class.

It is often called data hiding because it restricts direct access to a class's data members from outside the class. Instead, access to the data is controlled through methods, typically referred to as getters and setters.

Here's why encapsulation is called data hiding:

* **Access Control:**
* Encapsulation allows you to specify the visibility of data members and methods using access modifiers like public, private, protected, and default. By declaring data members as private, you prevent external code from directly accessing or modifying them.
* **Controlled Access:**
* Getters (accessor methods) and setters (mutator methods) provide controlled access to the data. You can add validation logic within setters to ensure that only valid data is assigned to the attributes.
* **Implementation Hiding:**
* Encapsulation hides the internal implementation details of a class from the outside world. This means you can change how the data is stored or processed internally without affecting the code that uses the class.

1. **What is the important feature of encapsulation?**

Ans: Encapsulation is a technique in object-oriented programming (OOP) that bundles data and methods into a single unit. It can be used to hide data and methods associated with an object, and has several benefits, including:

* Hiding data: Prevents users from knowing how classes are implemented or stored
* More flexibility: Allows variables to be set as read or write-only
* Easy to reuse: Makes it easier to change and adapt to new requirements
* Prevents unexpected modifications: Prevents external code from modifying data inside an object in a different part of the program
* Makes code more maintainable: Allows updates without changing input/output formats
* Improves readability and security: Groups data and methods together, and helps control the modification of data members

**3. what are the getter and setter methods in java with example?**

Ans:- In Java, getter and setter methods are used to access and modify the values of private fields within a class.

Getters

* Retrieve the value of a private field.
* Start with the prefix "get" followed by the variable name with the first letter capitalized.
* Return the value of the corresponding private field.

Setters

* Modify the value of a private field.
* Start with the prefix "set" followed by the variable name with the first letter capitalized.
* Take a parameter of the same type as the private field and assign it to the field.

Example:

public class Person {

private String name;

private int age;

// Getter for name

public String getName() {

return name;

}

// Setter for name

public void setName(String name) {

this.name = name;

}

// Getter for age

public int getAge() {

return age;

}

// Setter for age

public void setAge(int age) {

this.age = age;

}

}

**4. what are the getter and setter methods in java with example?**

**Ans : In Java, the this keyword refers to the current object within a method or constructor.**

**Here are a few common uses of the this keyword:**

* **Resolving Ambiguity: When a local variable or parameter has the same name as an instance variable, this is used to differentiate between the two. It explicitly refers to the instance variable.**

**Java**

**class MyClass {**

**int x;**

**public MyClass(int x) {**

**this.x = x; *// 'this.x' refers to the instance variable, 'x' refers to the parameter***

**}**

**}**

* **Calling Another Constructor: Within a constructor, this() can be used to invoke another constructor of the same class. This is useful for chaining constructors to avoid code repetition.**

**Java**

**class MyClass {**

**int x;**

**int y;**

**public MyClass(int x) {**

**this(x, 0); *// Calls the constructor below with 'x' and a default 'y' value***

**}**

**public MyClass(int x, int y) {**

**this.x = x;**

**this.y = y;**

**}**

**}**

* **Returning the Current Object: A method can return the current object using return this;. This technique is often used in builder pattern implementations for method chaining.**

**Java**

**class MyClass {**

**int x;**

**public MyClass setX(int x) {**

**this.x = x;**

**return this; *// Returns the current MyClass object***

**}**

**}**

**In summary, the this keyword in Java provides a way to explicitly reference the current object's members and enables several useful programming patterns.**

**5. what is the advantages of encapsulation in java?**

**Ans : Data Hiding:**

* **This is the core principle of encapsulation. By making the internal state of an object private and only accessible through controlled methods, you prevent unauthorized access and modification. This protects data integrity and ensures that objects are used in a predictable way.**
* **Increased Flexibility:**
* **You can modify the implementation details of a class without affecting the code that uses it. This is because the internal workings are hidden from the outside world.**
* **Code Reusability:**
* **Encapsulated classes can be used in different parts of your application or even in different projects. This promotes code reuse and saves development time.**
* **Improved Maintainability:**
* **Encapsulation makes code easier to understand and maintain. Since each object is responsible for its own data and behavior, it's easier to locate and fix issues.**
* **Enhanced Security:**
* **By restricting access to data, encapsulation helps prevent unauthorized modifications and ensures that objects remain in a valid state.**

**6. how to achieve encapsulation in java?**

**Ans :Encapsulation in Java is achieved by:**

* **Declaring variables as private: This prevents direct access to the variables from outside the class.**
* **Providing public getter and setter methods: These methods allow controlled access to the private variables.**

**public class Person {**

**private String name;**

**private int age;**

**// Getter for name**

**public String getName() {**

**return name;**

**}**

**// Setter for name**

**public void setName(String name) {**

**this.name = name;**

**}**

**// Getter for age**

**public int getAge() {**

**return age;**

**}**

**// Setter for age**

**public void setAge(int age) {**

**if (age >= 0) { // Example of data validation in setter**

**this.age = age;**

**} else {**

**System.out.println("Age cannot be negative.");**

**}**

**}**

**}**