1) What is encapsulation in Java? Why is it called Data hiding?

Ans:

Encapsulation :

- Encapsulation in Java is a fundamental principle of object-oriented programming (OOP).

- It involves bundling data (variables) and methods (functions) together into a single unit called a class.

- Encapsulation protects the integrity and consistency of data by declaring data members as private, preventing direct access from outside the class.

- Access to the encapsulated data is provided through public methods (getters and setters) that control and validate the access and modification of the data.

- Encapsulation promotes code organization by grouping related data and methods within a class.

Why it is called data hiding?

1. Concealing Implementation Details: Encapsulation encapsulates the data within a class, making it private or inaccessible from outside the class. This hides the internal implementation details of the data structure, such as its representation, organization, or any related algorithms. By hiding these implementation details, the class can maintain control over how the data is accessed and modified.

2. Limited Access via Methods: Encapsulation provides controlled access to the encapsulated data through public methods (getters and setters). These methods act as intermediaries, allowing external code to interact with the data within the predefined boundaries set by the class. The class can enforce any necessary rules, validations, or transformations within these methods, maintaining the integrity and consistency of the data.

3. Focus on Interface, Not Implementation: Encapsulation promotes the idea of treating an object as a black box, where users of the class are only concerned with its public interface. By hiding the internal details, such as the data structure or algorithms used, the users can focus on using the class without worrying about how the data is stored or processed internally. This abstraction of the implementation details simplifies the usage and understanding of the class.

2) What are the important features of encapsulation?

Ans:

Features of encapsulation:-
1. Data Hiding: Encapsulation hides the internal data of a class, making it private and inaccessible from outside. This ensures data integrity and prevents unauthorized modifications, as external code cannot directly access or modify the encapsulated data.

2. Access Control: Encapsulation provides controlled access to data through public methods (getters and setters). These methods act as an interface to interact with the encapsulated data, enforcing rules, validations, and maintaining data consistency. They allow for read-only or write-only access, and can perform additional operations as needed.

3. Abstraction: Encapsulation separates the external interface of a class from its internal implementation. External code interacts with the class through the public interface, without needing knowledge of the underlying details. This simplifies usage, promotes code modularity, and abstracts away complexity, allowing users to focus on what the class does rather than how it does it.

4. Data Bundling: Encapsulation bundles related data and methods into a single unit called a class. It allows for the organization of data and associated behavior within a cohesive entity. The class encapsulates the state and behavior of an object, providing a clear representation of real-world entities or concepts.

5. Code Flexibility: Encapsulation enables modifications to the internal implementation of a class without affecting the external code. As long as the public interface remains unchanged, internal changes, such as optimizations or enhancements, can be made seamlessly. This ensures easier maintenance, code evolution, and promotes code flexibility.

3) What are getter and setter methods in Java Explain with an example.

Ans:

Getter Method:
A getter method, also known as an accessor method, is a public method in a class that retrieves and returns the value of a private data member (variable). It provides indirect access to the private data member, allowing external code to obtain the current value of the variable. Getter methods typically have a return type and do not take any parameters.

Setter Method:

A setter method, also known as a mutator method, is a public method in a class that sets or modifies the value of a private data member (variable). It provides indirect access to the private data member, allowing external code to update the value of the variable. Setter methods typically do not have a return type but take one or more parameters representing the new value(s) to be assigned to the variable.

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) {
        if (age >= 0) {
            this.age = age;
        } else {
            System.out.println("Invalid age value. Age must be non-negative.");
        }
    }
}
```

4) What is the use of this keyword explain with an example.

Ans:
The `this` keyword in Java is used as follows:

1. Referencing Current Object: The primary use of `this` is to refer to the current object instance within a class. It can be used to access the instance variables and methods of the current object.

2. Differentiating Between Local and Instance Variables: When a local variable or method parameter has the same name as an instance variable, the `this` keyword is used to differentiate between them. It allows you to explicitly refer to the instance variable rather than the local variable or parameter.

3. Invoking Constructors: The `this` keyword can be used to invoke one constructor from another constructor within the same class. It enables constructor chaining, where one constructor can call another constructor to initialize the object.

4. Returning the Current Object: In certain cases, a method can return the current object itself. By using the `this` keyword, you can return the reference to the current object from within the method.

Example that demonstrates these uses of the `this` keyword:

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public Person getPerson() {
        return this;
    }
}


public class Main {
    public static void main(String[] args) {
        Person person1 = new Person("John");
```

```
        Person person2 = new Person("Alice");

        person1.setName("John Doe");

        System.out.println("Person 1: " + person1.getName());
        System.out.println("Person 2: " + person2.getName());

        Person person3 = person1.getPerson();
        System.out.println("Person 3: " + person3.getName());
    }
}
```

5) What are the advantages of encapsulation?

Ans:

Advantages of encapsulation:

1. Data Protection: Encapsulation hides the internal data of a class, making it private and inaccessible from the outside. This prevents unauthorised access and modifications, ensuring data integrity and security.

2. Code Modularity and Reusability: Encapsulation allows for the creation of self-contained and reusable components. Encapsulated classes can be easily integrated into different parts of a program or reused across multiple projects, improving code organization, maintainability, and reducing redundancy.

3. Controlled Access and Abstraction: Encapsulation provides controlled access to data through public methods (getters and setters). This allows the class to enforce rules, and validations, and perform necessary operations on the data. Encapsulation also abstracts away the internal implementation details, simplifying usage and reducing complexity.

4. Code Flexibility and Maintainability: Encapsulation enables changes to the internal implementation of a class without affecting the external code that uses it. By maintaining a stable public interface, internal modifications can be made transparently, improving code flexibility and maintainability. Encapsulation reduces dependencies, isolates change within the class and allow for easier bug fixes or enhancements.

5. Team Collaboration and Scalability: Encapsulation promotes team collaboration in software development projects. Well-encapsulated classes provide clear interfaces, making it easier for multiple developers to work on different parts of the code simultaneously. Encapsulation also facilitates scalability, as new features can be added to a system without impacting existing code, reducing conflicts and unintended side effects.

6) How to achieve encapsulation in Java? Give an example.

Ans:

To achieve encapsulation in Java, we can follow these steps:

1. Declare the Data Members as Private: Make the data members of a class private. This ensures that the data can only be accessed and modified from within the class itself.

2. Provide Public Getter and Setter Methods: Create public getter and setter methods to access and modify the private data members. These methods provide controlled access to the data and allow for any necessary logic or validations.

3. Use Access Modifiers: Use appropriate access modifiers (such as private, public, protected) to control the accessibility of the methods and data members. Private access ensures that the data members can only be accessed within the class, while public access allows external code to interact with the class through the provided methods.

4. Implement Validation and Logic in Setter Methods: In the setter methods, you can add checks and conditions to enforce rules and validations on the input values before updating the data members. This ensures that the data remains consistent and valid.

Here's an example that demonstrates encapsulation in Java:

```
public class Person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        name = newName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int newAge) {
```

```
        if (newAge >= 0) {
            age = newAge;
        } else {
            System.out.println("Invalid age value. Age must be non-negative.");
        }
    }
}


public class Main {
    public static void main(String[] args) {
        Person person = new Person();

        person.setName("John");
        System.out.println("Name: " + person.getName());

        person.setAge(25);
        System.out.println("Age: " + person.getAge());
    }
}
```