1. **What is Encapsulation in Java? Why is it called Data hiding?**

**Ans.** Binding of data and corresponding methods into a single unit is called “Encapsulation”.

Every member inside the class should be declared as private, and to access this private date we need to have setter and getter methods.

By using private modifiers we can implement “data hiding”.

1. **What are the important features of Encapsulation?**

**Ans.** i) we can achieve security

ii) Easy to improve

iii) Easy to maintain

iv) Code reusability

1. **What are getter and setter methods in Java Explain with an example?**

**Ans.**

**Setter methods are used to set the value to the instance variables of the class.**

Syntax for setter method:

a. compulsory the method name should start with set.

b. it should be public.

c. return type should be void.

d. compulsorily it should have some argument.

**Getter methods are used to get the value from the instance variables of the class.**

Syntax for getter method:

a. compulsory the method name should start with get.

b. it should be public.

c. return type should not be void.

d. compulsorily it should not have any argument.

1. **What is the use of this keyword explain with an example.**

**Ans.** “This” keyword would always point to current object, and “This” variable would hold the address the active object present in the heap memory.

Example:

public class Student{

                private boolean married;

                public void setMarried(Boolean married){

                     this.married=married;

              }

public boolean isMarried()(){

return married;

               }

}

1. **What is the advantage of Encapsulation?**

**Ans.**

1. Data Protection: Encapsulation ensures that data is protected from direct access and modification by external entities.
2. Modularity: Encapsulation promotes modularity by grouping related data and methods into a single unit, known as a class. Modularity enhances code readability, reusability, and maintainability, as changes made to one encapsulated class are less likely to impact other parts of the program.
3. Code Reusability: Encapsulation supports code reusability by encapsulating reusable functionality within a class. Once encapsulated, an object can be instantiated and used in multiple parts of a program without duplicating code.
4. Information Hiding: Encapsulation hides the internal implementation details of a class and provides a well-defined interface to interact with the object. This helps to manage the complexity of the codebase and reduces dependencies between different parts of the program.
5. Security and Access Control: Encapsulation allows for fine-grained control over access to data and methods. By marking certain members as private or protected, encapsulation prevents unauthorized access and modification from external classes. This provides a level of security and helps maintain data integrity. Access to encapsulated data is only allowed through public methods, enabling the class to enforce necessary checks and restrictions.
6. Evolution and Maintenance: Encapsulation makes it easier to modify and maintain code over time. The internal implementation details of a class can be changed without affecting other parts of the program that rely on the class. This concept is known as "encapsulation provides a stable interface," meaning that the external contract (public methods) remains consistent even if the internal implementation changes. This flexibility allows for more agile development, easier bug fixing, and seamless evolution of the codebase.
7. **How to achieve encapsulation in Java? Give an example.**

**Ans.** Encapsulation is achieved by using access modifiers (such as private, protected, and public) to control the visibility and accessibility of class members (fields and methods).

E.g.

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;

}

}

}

public class Main {

public static void main(String[] args) {

Person person = new Person();

person.setName("John");

person.setAge(25);

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

System.out.println("Age: " + person.getAge());

}

}