Assignment – 16th (Encapsulation)

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

Ans: - Encapsulation in Java is a concept of wrapping related data and methods into a single unit, known as a class. It is an important mechanism for achieving data hiding in object-oriented programming, which means that the implementation details of a class are kept hidden from the outside world.

Encapsulation in Java can be achieved through access modifiers such as public, private, and protected, which are used to control the visibility of the data and methods of a class. By making data members private and providing public methods to access and modify them, encapsulation ensures that the state of an object is only accessible through its defined methods.

Encapsulation is called data hiding because it allows us to hide the internal details of a class from external classes. This helps to prevent accidental modification of the data and promotes the idea of information hiding in object-oriented programming, where we aim to reduce the complexity of software systems by hiding the implementation details of a class from other classes.

2. What are the important features of Encapsulation?

Ans: - Encapsulation is an important concept in Java that helps to create secure and reliable software systems by hiding the implementation details of a class and controlling access to its data and methods. The important features of Encapsulation in Java are:

- Encapsulation helps to hide the internal details of a class and make it accessible only through defined methods. This makes the class easier to use and understand.
- Encapsulation uses access modifiers such as public, private, and protected to define the visibility of the data and methods of a class. This allows us to control the access to the data and methods and ensures the security and integrity of the class.
- Encapsulation is achieved through the use of classes in Java. A class is a blueprint that defines the data and methods that will be encapsulated.
- Encapsulation allows us to validate the data before it is stored in the data members of a class. This ensures that the data is consistent and valid.
- Encapsulation provides getter and setter methods to access and modify the data members of a class. These methods ensure that the data is accessed and modified in a safe and controlled way.
- Encapsulation promotes code reusability by allowing the encapsulated data and methods to be used in other classes without exposing their internal details.

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

Ans: - Getter and setter methods are methods used in Java to access and modify the private data members of a class. Getter methods are used to retrieve the value of a private data member, while setter methods are used to modify the value of a private data member. Getter and setter methods can be used to validate the data being accessed or modified.

Getter and setter methods are an important part of object-oriented programming in Java, and they can be used to implement encapsulation, validate data, calculate properties, and more.

When you declare a data member as private, it cannot be accessed from outside the class. This is a fundamental aspect of encapsulation in object-oriented programming, which helps to ensure that the implementation details of a class are hidden from the outside world. However, in some cases, you may want to allow controlled access to these private data members. To enable controlled access to private data members, you can define public getter and setter methods for each data member. The getter method allows you to retrieve the value of the data member, while the setter method allows you to modify the value of the data member. Here is an example of a class with private data members and corresponding getter and setter methods:

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

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

   // Setter method for name
   public void setName(String name) {
      this.name = name;
   }

   // Getter method for age
   public int getAge() {
      return age;
   }

   // Setter method for age
   public void setAge(int age) {
      this.age = age;
   }
}

public class Main {
   public static void main(String[] args) {
      Person person = new Person();
      person.setName("Rahul Kumar");
      person.setAge(20);

      System.out.println("Name: " + person.getName());
      System.out.println("Age: " + person.getAge());
   }
}
```

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

Ans: - In Java, the `***this***` keyword is used to refer to the current instance of a class. It can be used to access or modify the current object's data members or methods. Inside the constructor, we use the `***this***` keyword to refer to the current object's data members, since the parameters to the constructor have the same names as the data members. Without the `***this***` keyword, the constructor would assign the parameter values to themselves instead of the data members. Here is an example:

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

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

   public void printDetails() {
      System.out.println("Name: " + this.name);
      System.out.println("Age: " + this.age);
   }
}
```

5. What is the advantages of Encapsulation?

Ans: - Encapsulation is a powerful technique in Java that helps to create well-organized, maintainable, and secure code. By hiding internal details, promoting modularity and reusability, providing flexibility, and enhancing security, encapsulation helps to create more robust and reliable software. It is an important concept and has several advantages, including:

- Encapsulation provides a way to hide the internal details of a class from the outside world. This means that the data members of a class can be made private, and access to them can be controlled through public methods. This helps to prevent unwanted modifications to the data and ensures that the class is used in the way it was intended.
- Encapsulation provides a way to change the internal implementation of a class without affecting the external interface. This means that improvements can be made to a class without breaking existing code that uses it. Additionally, encapsulation provides a way to add new features to a class without affecting the existing code.
- Encapsulation provides a way to protect sensitive data by hiding it from unauthorized access. By making data members private and providing public methods to access them, access to the data can be controlled and restricted to authorized users.
- Encapsulation provides a way to improve the performance of a program by controlling access to data members. By making data members private, access can be controlled and optimized through methods. This can improve the efficiency of the program by reducing unnecessary access to data, and improving cache utilization.
- Encapsulation provides a way to optimize code by reducing the amount of duplicate code.

6. How to achieve encapsulation in Java, Give an Example?

Ans: - Encapsulation in Java can be achieved by declaring the class's data members as private and providing public methods (also known as accessor methods) to access and modify them. This prevents direct access to the data members from outside the class and allows for better control over their use. By using encapsulation, we can ensure that the data members of the class are only accessed and modified through the defined public methods. This makes the code easier to maintain and less prone to errors, and allows for better control over the use of the class's data members.

Here's an example of encapsulation in Java:

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

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

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

}

public class Main {
    public static void main(String[] args) {

        Person person = new Person("Rahul Kumar", 20);
        String name = person.getName();
        int age = person.getAge();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```