Abstraction and encapsulation are part of the Object-Oriented Programming paradigm that also includes classes, inheritance, objects, and polymorphism. This paradigm reduces complexity and increases a program's maintainability by utilizing abstraction and encapsulation. (Liang, Introduction to Java Programming and Data Structures, p. 397).

The Object-Oriented Programming paradigm also promotes modularity and reusability as its advantages.

In Java, encapsulation binds data and methods together as a single unit, which helps us keep related fields and methods together while making the code legible. The data fields of a class can be protected using access modifiers such as public, private, protected, and default to restrict access. Restricted data fields can only be accessed or modified by the designated accessor and mutator methods. This method of encapsulating is known as data hiding.

Class Abstraction and Method abstraction both separate and encapsulate their implementation and public interface. Therefore, other classes or users cannot directly affect the inner workings and must use the public interface of the class or method. Abstraction empowers modular programs that are hierarchically structured and are easier to write, debug, maintain, and modify (Liang, Introduction to Java Programming and Data Structures, p. 257). Abstraction is achievable using interfaces and abstract classes mechanisms that play an influential role in avoiding code duplication and allows for reusability. Abstract classes can allow you to implement similar concrete and non-concrete methods across multiple common classes. Interfaces can use different classes that share common methods and have different implementation details by using polymorphism to reduce code duplication. To sum up, abstraction and encapsulation promotes modular code that is easier to write, debug, maintain, modify, and reuse.