Using OOP concepts to write high-performance Java code

By Anna Monus | Posted Feb 23, 2022 | 13 min. (2710 words)

Java is a class-based object-oriented programming (OOP) language built around the concept of objects. OOP concepts are intended to improve code readability and reusability by defining how to structure your Java program efficiently. The core principles of object-

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism
5. Association
6. Aggregation
7. Composition

Java comes with specific code structures for each OOP concept, such as the `extends` keyword for the inheritance principle or the getter and setter methods for the encapsulation principle.

While these concepts are crucial for creating well-structured Java programs in the development phase, implementing

1. Abstraction

Abstraction aims to hide complexity from users and show them only relevant information. For example, if you're driving a car, you don't need to know about its internal workings.

The same is true of Java classes. You can hide internal implementation details using abstract classes or interfaces. On the abstract level, you only need to define the method signatures (name and parameter list) and let each class implement them in their own way.

2. Encapsulation

Encapsulation helps with data security, allowing you to protect the data stored in a class from system-wide access. As the name suggests, it safeguards the internal contents of a class like a capsule.

You can implement encapsulation in Java by making the fields (class variables) private and accessing them via their public getter and setter methods. JavaBeans are examples of fully encapsulated classes.

3. Inheritance

Inheritance makes it possible to create a child class that inherits the fields and methods of the parent class. The child class can override the values and methods of the parent class, but it's not necessary. It can also add new data and functionality to its parent.

Parent classes are also called superclasses or base classes, while child classes are known as subclasses or derived classes as well. Java uses the `extends` keyword to implement the

Polymorphism refers to the ability to perform a certain action in different ways. In Java, polymorphism can take two forms: method overloading and method overriding.

Method overloading happens when various methods with the same name are present in a class. When they are called, they are differentiated by the number, order, or types of their parameters. Method overriding occurs when a child class overrides a method of its parent.

5. Association

Association means the act of establishing a relationship between two unrelated classes. For example, when you declare two fields of different types (e.g. `Car` and `Bicycle`) within the same class and make them interact with each other, you have created an association.

Association in Java:

- Two separate classes are associated through their objects
- The two classes are unrelated, each can exist without the other one
- Can be a one-to-one, one-to-many,

6. Aggregation

Aggregation is a narrower kind of association. It occurs when there's a one-way (HAS-A) relationship between the two classes we associate through their objects.

For example, every `Passenger` has a `Car`, but a `Car` doesn't necessarily have a `Passenger`. When you declare the `Passenger` class, you can create a field of the `Car` type that shows which car the passenger belongs to. Then, when you instantiate a new `Passenger` object, you can access the data stored in the related `Car` as well.

7. Composition

Composition is a stricter form of aggregation. It occurs when the two classes you associate are mutually dependent and can't exist without each other.

For example, take a `Car` and an `Engine` class. A `Car` cannot run without an `Engine`, while an `Engine` also can't function without being built into a `Car`. This kind of relationship between objects is also called a PART-OF relationship.