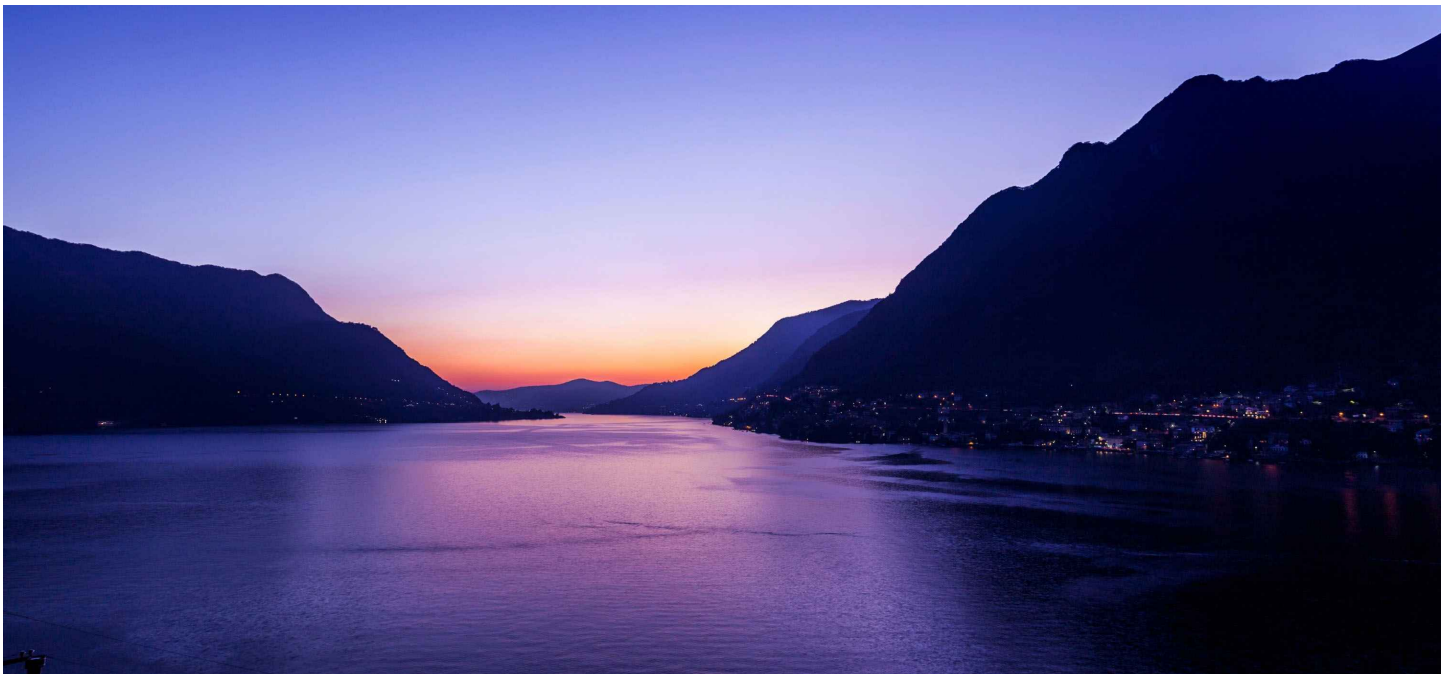

20-Object Oriented Programming

Here are some examples to illustrate the difference between object-oriented programming (OOP) and non-OOP approaches using the Java programming language:

Example of OOP

```java
public class Car {
    private String brand;
    private String model;
    private int year;

    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    public void startEngine() {
        // Code to start the car's engine
    }

    public void accelerate() {
        // Code to accelerate the car
    }
}
```

In this example, we have a `Car` class that encapsulates data (brand, model, year) and behavior (startEngine(), accelerate()). The data and methods related to a car are bundled together in the class, promoting code **modularity** and **reusability**.

Example of Non-OOP (Procedural Programming)

```java
public class Car {
    private String brand;
    private String model;
    private int year;
}

public class Forever {

    // Car Behavior
    public static void startCarEngine(Car car) { // pass by reference
        // Code to start the car's engine

    }
    // Car Behavior
    public static void accelerateCar(Car car) { // pass by reference
        // Code to accelerate the car
    }

    public static void createBall() {
    }

    public static void changeBallColor(Ball ball) {
        ball.setColor("red");
    }
}
```

In this example, the **data and behavior related to a car are separated**. We have a `Car` class that only encapsulates data, and the behavior is defined as standalone functions (`startCarEngine()` and `accelerateCar()`) that take a `Car` object as a parameter

Example of OOP (Inheritance and Polymorphism)

```java
public abstract class Shape {
    public abstract double calculateArea();
```

```java
}

public class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}

public class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    public double calculateArea() {
        return length * width;
    }
}
```

- In the above example, we have an `abstract Shape` class representing a generic shape with an abstract method `calculateArea()`. The `Circle` and `Rectangle` classes inherit from `Shape` and provide their own implementation of the `calculateArea()` method.
- This demonstrates the use of **inheritance** and **polymorphism**, where objects of different types can be treated uniformly through a common interface (`Shape`).

Summary

To conclude, all the above examples aims to highlight the differences between object-oriented programming (OOP), where data and behavior are encapsulated in classes, and non-OOP approaches, where data and behavior may be separate or organized differently.

OOP promotes code **modularity**, **reusability**, and **modeling real-world entities**, while non-OOP approaches may focus more on **procedural steps** or functional composition.