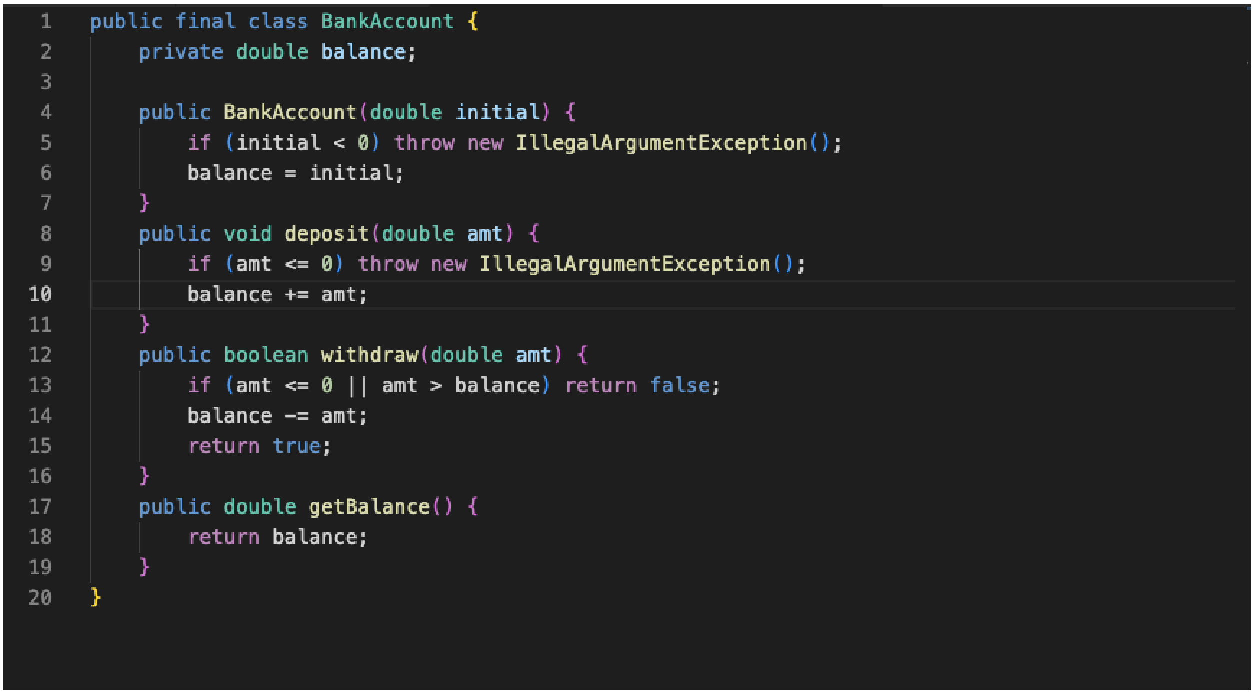
I chose Encapsulation and Polymorphism as a persuasive pair for OO fundamentals.

1. Encapsulation
   1- Encapsulation makes a class guarantee that its data always remain in a valid configuration by keeping state private and exposing only controlled accessors.
   2- A defined public interface works like a contract that other module can rely on which is the key in terms of improving communications.
   3- It reduces the risk and cost of maintenance of callers in large code bases by hiding implementation details.



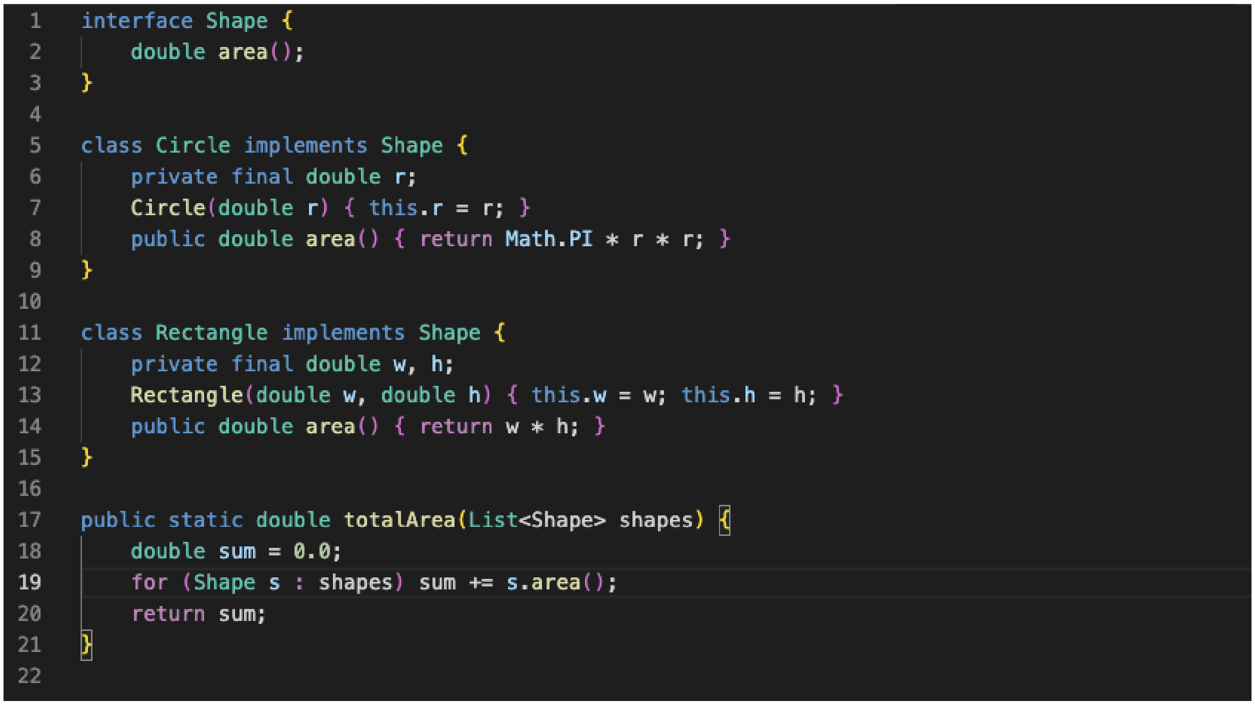
```java
public final class BankAccount {
    private double balance;

    public BankAccount(double initial) {
        if (initial < 0) throw new IllegalArgumentException();
        balance = initial;
    }
    public void deposit(double amt) {
        if (amt <= 0) throw new IllegalArgumentException();
        balance += amt;
    }
    public boolean withdraw(double amt) {
        if (amt <= 0 || amt > balance) return false;
        balance -= amt;
        return true;
    }
    public double getBalance() {
        return balance;
    }
}
```

2. Polymorphism

   1- It allows new behaviors to be added by adding new subclasses rather than editing existing code, which keeps tested code untouched.
   2- It leads to cleaner and shorter code since callers work with an abstract type and need not branch on concrete classes.

3- Runtime objects can be swapped without altering client code.



```java
interface Shape {
    double area();
}

class Circle implements Shape {
    private final double r;
    Circle(double r) { this.r = r; }
    public double area() { return Math.PI * r * r; }
}

class Rectangle implements Shape {
    private final double w, h;
    Rectangle(double w, double h) { this.w = w; this.h = h; }
    public double area() { return w * h; }
}

public static double totalArea(List<Shape> shapes) {
    double sum = 0.0;
    for (Shape s : shapes) sum += s.area();
    return sum;
}

```