Task 3: Functional Interfaces
Create a method that accepts functions as parameters using Predicate, Function, Consumer, and Supplier interfaces to operate on a Person object.
ANS:

```
import java.util.function.Predicate;
import java.util.function.Function;
import java.util.function.Consumer;
import java.util.function.Supplier;

class Person {
    private String name;
    private int age;

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

    // Getters and Setters
    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;
    }

    // toString method for printing
    @Override
    public String toString() {
        return name + ": " + age;
    }

    // Method to demonstrate functional interfaces
    public static void operateOnPerson(
            Person person,
            Predicate<Person> predicate,
            Function<Person, String> function,
            Consumer<Person> consumer,
```

```
            Supplier<Person> supplier) {

        // Predicate: Test condition on person
        if (predicate.test(person)) {
            System.out.println("Predicate condition is true for: " + person);
        } else {
            System.out.println("Predicate condition is false for: " + person);
        }

        // Function: Apply function to person
        String result = function.apply(person);
        System.out.println("Function result: " + result);

        // Consumer: Perform operation on person
        consumer.accept(person);
        System.out.println("After Consumer operation: " + person);

        // Supplier: Get a new person
        Person newPerson = supplier.get();
        System.out.println("Supplied new person: " + newPerson);
    }

    public static void main(String[] args) {
        Person person = new Person("Alice", 30);

        // Define functional interfaces
        Predicate<Person> isAdult = p -> p.getAge() >= 18;
        Function<Person, String> personToName = Person::getName;
        Consumer<Person> increaseAge = p -> p.setAge(p.getAge() + 1);
        Supplier<Person> newPersonSupplier = () -> new Person("Bob", 25);

        // Operate on person using functional interfaces
        operateOnPerson(person, isAdult, personToName, increaseAge, newPersonSupplier);
    }
}
```