SOLID Principles: Assignment — Interface Segregation Principle (ISP) in a Device Suite

**Objective** Prove you can spot a monolithic interface, split it into focused contracts, refactor implementations, and document the design pattern you applied.  
 **Starter code (keep in src/main/java/legacy/)**

package legacy;

public interface Machine {

void start();

void stop();

void print(String document);

}

package legacy;

public class Printer implements Machine {

public void start(){ System.out.println("Printer starting..."); }

public void stop(){ System.out.println("Printer stopping..."); }

public void print(String d){ System.out.println("Printing: "+d); }

}

package legacy;

public class CoffeeMachine implements Machine {

public void start(){ System.out.println("Coffee machine starting..."); }

public void stop(){ System.out.println("Coffee machine stopping..."); }

public void print(String d){ throw new UnsupportedOperationException("Cannot print"); }

}

**Tasks** 1 Analyse the violation: list every unused or harmful method each concrete class inherits in analysis/isp\_problems.md and include a code snippet showing the runtime failure.  
 2 Write a failing JUnit test (CoffeeMachineShouldNotPrintTest) that asserts UnsupportedOperationException is thrown when print is called on CoffeeMachine.  
 3 Refactor under src/main/java/clean/  
 • Create BasicDevice (start/stop) and Printable (print) interfaces.  
 • Implement Printer implements BasicDevice, Printable and CoffeeMachine implements BasicDevice.  
 • Create a new class Scanner that implements BasicDevice, Printable (prints “scanned to PDF …”).  
 4 Demonstrate polymorphism in Main.java: store List<BasicDevice> and call only start/stop; store List<Printable> and call print.  
 5 Reflection in reflection.md  
 • How did the original interface break ISP?  
 • Which architecture pattern did you apply (answer: *Interface Segregation via role-based interfaces*)?  
 • What new flexibility did the smaller interfaces unlock?

**Deliverables**

analysis/isp\_problems.md

src/main/java/clean/\*\* ← new interfaces & classes

src/test/java/\*\* ← failing then passing tests

src/main/java/Main.java ← demo lists

reflection.md

README.md ← build & run instructions

**Solution (clean module)**

package clean;

public interface BasicDevice { void start(); void stop(); }

package clean;

public interface Printable { void print(String document); }

package clean;

public class Printer implements BasicDevice, Printable {

public void start(){ System.out.println("Printer starting..."); }

public void stop(){ System.out.println("Printer stopping..."); }

public void print(String d){ System.out.println("Printing: "+d); }

}

package clean;

public class CoffeeMachine implements BasicDevice {

public void start(){ System.out.println("Coffee machine starting..."); }

public void stop(){ System.out.println("Coffee machine stopping..."); }

}

package clean;

public class Scanner implements BasicDevice, Printable {

public void start(){ System.out.println("Scanner starting..."); }

public void stop(){ System.out.println("Scanner stopping..."); }

public void print(String d){ System.out.println("Scanned to PDF: "+d); }

}

package clean;

import java.util.\*;

public class Main {

public static void main(String[] args) {

List<BasicDevice> devices = Arrays.asList(new Printer(), new CoffeeMachine(), new Scanner());

devices.forEach(BasicDevice::start);

List<Printable> outputs = Arrays.asList(new Printer(), new Scanner());

outputs.forEach(p -> p.print("ISP Handbook"));

}

}

**Concept recap in solution** The refactor illustrates ISP by splitting one broad interface (Machine) into two minimal contracts (BasicDevice, Printable). Clients now depend only on the behaviours they use, implementations avoid empty or exception-throwing methods, and new devices can mix capabilities through composition of interfaces.