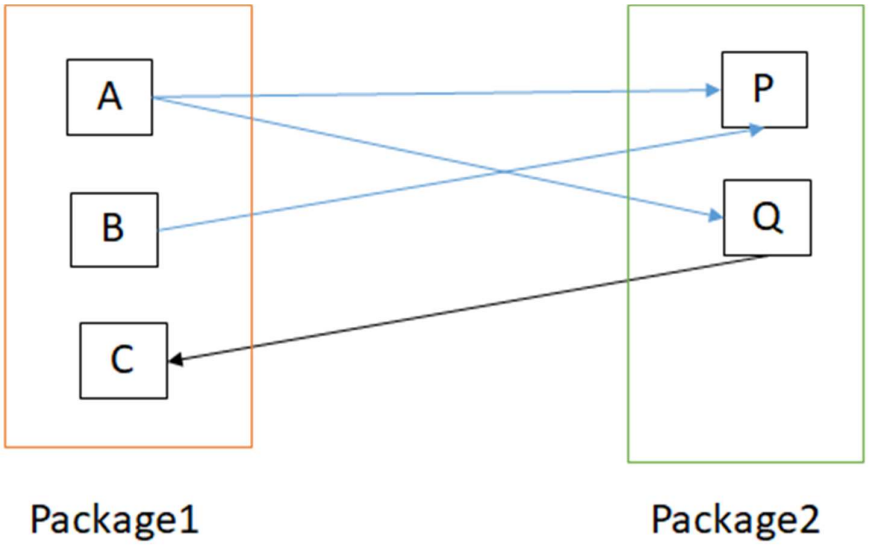
Submitted by - Rithika

Afferent Couplings (Ca): The number of classes in other packages that depend upon classes within the package is an indicator of the package's responsibility. Afferent = incoming

- A class afferent couplings is a measure of how many other classes use the specific class.

Efferent Couplings (Ce): The number of classes in other packages that the classes in the package depend upon is an indicator of the package's dependence on externalities. Efferent = outgoing.

- A class efferent couplings is a measure of how many different classes are used by the specific class.



Package1,

- class **A** has 2 out-going edges,
- class **B** has 1 out-going edge,
- class **C** has 1 in-coming edge
- Calculation:

  Ca = {} union {} union {Q}
  = {Q}
  = 1

  Ce = {P, Q} union {P} union {}
  = {P, Q,} =2

Package2,

- class **P** has 2 in-coming edges,
- class **Q** has 1 in-coming and 1 out-going edge

  Ca = {A, B} union {A}
  = {A, B}
  = 2
  Ce = {} union {C}
  = {C}=1

Instability is the ratio between Efferent Coupling (Ce) and the total package coupling (Ce + Ca)

I = Ce / (Ce + Ca) =[0,1] 0-> Unstable, 1->Stable

IN **Java:**

AuthSystem.java

```java
// AuthSystem.java
public class AuthSystem {
    public AuthSystem() {
        System.out.println("AuthSystem created");
    }
}
```

Database.java

```java
// Database.java
public class Database {
    public Database() {
        System.out.println("Database created");
    }
}
```

Emailer.java

```java
// Emailer.java
public class Emailer {
    public Emailer() {
        System.out.println("Emailer created");
    }
}
```

Program.java

```java
// Program.java
public class InstabilityDemo {
    public static void main(String[] args) {
        new Coordinator().doWork();
    }
}
```

```
> EI exercise > 5Days-SoftwareArchitecture > Java > InstabilityDemo > J Coor
1   // Coordinator.java
2   public class Coordinator {
3       public void doWork() {
4           AuthSystem a = new AuthSystem(); // Ce += 1
5           Database b   = new Database();   // Ce += 1
6           Emailer c    = new Emailer();    // Ce += 1
7           System.out.println("Coordinator working...");
8       }
9   }
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\InstabilityDemo>javac *.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\InstabilityDemo>java InstabilityDemo
AuthSystem created
Database created
Emailer created
Coordinator working...
```

IN TYPESCRIPT:

```
EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > InstabilityD
    // Coordinator.ts
    import { AuthSystem } from "./Auth";    // Ce += 1
    import { Database } from "./DB";        // Ce += 1
    import { Emailer } from "./Email";      // Ce += 1

    export class Coordinator {
        doWork(): void {
            new AuthSystem();
            new Database();
            new Emailer();
            console.log("Coordinator working...");
        }
    }
```

```
D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Instabili
1   // instabilitydemo.ts
2   import { Coordinator } from "./Coordinator";
3
4   new Coordinator().doWork();
```

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark >

```ts
// Email.ts
export class Emailer {
    constructor() {
        console.log("Emailer created");
    }
}
```

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > In

```ts
// DB.ts
export class Database {
    constructor() {
        console.log("Database created");
    }
}
```

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > In

```ts
// Auth.ts
export class AuthSystem {
    constructor() {
        console.log("AuthSystem created");
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\InstabilityDemo>tsc instabilitydemo
.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\InstabilityDemo>node instabilitydem
o.js
AuthSystem created
Database created
Emailer created
Coordinator working...
```

Java coupling = using the class
TypeScript coupling = importing the module

Coding assessment:

Dependency Grapher:

Ex (A ---> B ---> C)	O/p:
A.java imports B	A.java -> B
B.java imports C	B.java -> C
C.java imports none	C.java -> (no dependencies)

In Java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > DependencyGrapherSimple.java

```java
import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.util.stream.*;

public class DependencyGrapherSimple {
    public static void main(String[] args) throws IOException {
        Path root = Paths.get(args.length > 0 ? args[0] : ".");
        Map<String, Set<String>> graph = new TreeMap<>();

        Files.walk(root)
             .filter(p -> p.toString().endsWith(".java"))
             .forEach(p -> {
                 String rel = root.relativize(p).toString().replace(File.separatorChar, '/');
                 graph.putIfAbsent(rel, new TreeSet<>());
                 try (BufferedReader r = Files.newBufferedReader(p)) {
                     String line;
                     while ((line = r.readLine()) != null) {
                         line = line.trim();
                         if (line.startsWith("import ")) {

                             String imp = line.substring(7).replace(";", "").trim();
                             graph.get(rel).add(imp);
                         }
                     }
                 } catch (IOException e) {
                     System.err.println("Read error: " + p + " -> " + e.getMessage());
                 }
             });

        // print adjacency list
        System.out.println("Java imports adjacency:");
        graph.forEach((from, tos) -> {
            if (tos.isEmpty()) return;
            System.out.println(from + " ->");
            tos.forEach(to -> System.out.println("  - " + to));
        });
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java>javac DependencyGrapherSimple
.java

D:\EI exercise\5Days-SoftwareArchitecture\Java>java DependencyGrapherSimple
Java imports adjacency:
DependencyGrapherSimple.java ->
  - java.io.*
  - java.nio.file.*
  - java.util.*
  - java.util.stream.*
```

In Typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > ts-dependency-grapher > TS

```typescript
import * as ts from "typescript";
import * as fs from "fs";
import * as path from "path";

interface FileDependencies {
    file: string;
    imports: string[];
}

// Recursively collect all .ts files in a folder
function getAllTsFiles(dir: string): string[] {
    let results: string[] = [];
    const list = fs.readdirSync(dir);
    for (const file of list) {
        const fullPath = path.join(dir, file);
        const stat = fs.statSync(fullPath);
        if (stat && stat.isDirectory()) {
            results = results.concat(getAllTsFiles(fullPath));
        } else if (fullPath.endsWith(".ts")) {
            results.push(fullPath);
        }
    }
    return results;
}

// Parse a single TypeScript file and collect imports
function getImports(filePath: string): string[] {
    const sourceCode = fs.readFileSync(filePath, "utf-8");
    const sourceFile = ts.createSourceFile(
        filePath,
        sourceCode,
        ts.ScriptTarget.Latest,
        true
    );

    const imports: string[] = [];
```

```
    function visit(node: ts.Node) {
        if (ts.isImportDeclaration(node)) {
            const moduleSpecifier = (node.moduleSpecifier as ts.StringLiteral).text;
            imports.push(moduleSpecifier);
        }
        ts.forEachChild(node, visit);
    }

    visit(sourceFile);
    return imports;
}

// Main function
function analyzeDependencies(folder: string): FileDependencies[] {
    const files = getAllTsFiles(folder);
    const result: FileDependencies[] = [];

    for (const file of files) {
        const imports = getImports(file);
        result.push({ file: path.relative(folder, file), imports });
    }

    return result;
}

// Usage
const folderToAnalyze = "./"; // current folder
const dependencies = analyzeDependencies(folderToAnalyze);

console.log("=== Dependency Graph ===");
dependencies.forEach(dep => {
    console.log(`${dep.file}:`);
    dep.imports.forEach(i => console.log(`  -> ${i}`));
});
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\ts-dependency-grapher
>npx ts-node dependencyGrapher.ts
=== Dependency Graph ===
dependencyGrapher.ts:
  -> typescript
  -> fs
  -> path
node_modules\@cspotcode\source-map-support\register-hook-require.d.ts:
node_modules\@cspotcode\source-map-support\register.d.ts:
node_modules\@cspotcode\source-map-support\source-map-support.d.ts:
node_modules\@jridgewell\resolve-uri\dist\types\resolve-uri.d.ts:
node_modules\@jridgewell\sourcemap-codec\src\scopes.ts:
  -> ./strings
  -> ./vlq
node_modules\@jridgewell\sourcemap-codec\src\sourcemap-codec.ts:
  -> ./vlq
  -> ./strings
node_modules\@jridgewell\sourcemap-codec\src\strings.ts:
node_modules\@jridgewell\sourcemap-codec\src\vlq.ts:
  -> ./strings
node_modules\@jridgewell\trace-mapping\dist\types\any-map.d.ts:
  -> ./trace-mapping
  -> ./types
node_modules\@jridgewell\trace-mapping\dist\types\binary-search.d.ts:
  -> ./sourcemap-segment
node_modules\@jridgewell\trace-mapping\dist\types\by-source.d.ts:
  -> ./sourcemap-segment
  -> ./binary-search
```

Fragility of inheritance?

A piece of code that easily breaks when you change something small somewhere else — even if the change should not logically affect it.

Why is below program fragile?

CountedList depends on the assumption that addRange() will always call add() once per item.

In java:

D: > El exercise > 5Days-SoftwareArchitecture > Java > Fragile > FragileInheritanceExample.java

```java
import java.util.Arrays;
import java.util.List;

class ListManager {


    public void addRange(List<String> items) {
        for (String item : items) {
            add(item); // virtual call
        }
    }

    public void add(String item) {
        // base add logic
        // (simulated empty for demo)
    }
}

class CountedList extends ListManager {

    public int count = 0;

    @Override
    public void add(String item) {
        count++;                    // Counting how many items were added
        super.add(item);            // Still call base logic
    }
}

public class FragileInheritanceExample {
    public static void main(String[] args) {
        CountedList list = new CountedList();
        list.addRange(Arrays.asList("a", "b"));

        System.out.println("Count = " + list.count);
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Fragile>javac FragileInherita
nceExample.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Fragile>java FragileInheritan
ceExample
Count = 2
```

In typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Fragility > TS FragileInheritanceE

```typescript
class ListManager {
    private items: string[] = [];
    add(item: string) { this.items.push(item); }
    // Fragile Method: Relies on internal call to 'add'
    addRange(items: string[]) { items.forEach(i => this.add(i)); }
}
class CountedList extends ListManager {
    count = 0;
    override add(item: string) {
        this.count++;
        super.add(item);
    }
}
// Usage
const list = new CountedList();
list.addRange(["a", "b"]);
console.log(list.count);
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Fragility>tsc  Fragil
eInheritanceExample.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Fragility>node  Fragi
leInheritanceExample
2
```

My Example:

Parent changes the car from manual to automatic

Child still expects manual → now it cannot drive

In Typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Fragility > TS

```ts
// Parent class
class Parent {
    greet() {
        console.log("Hello");
    }
}

// Child class
class Child extends Parent {
    greet() {
        super.greet(); // depends on Parent's greet
        console.log("Hi from Child");
    }
}

const c = new Child();
c.greet();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark>cd fragility

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Fragility>tsc child-parent
.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Fragility>node child-paren
t
Hello
Hi from Child
```

But if parent changes like :

```
class Parent {
   greet(name: string) {   // added a parameter
      console.log("Hello " + name);
   }
}
```

ERROR: greet expects 1 argument - >**Fragility**

In Java:

If Vehicle.drive() changes its parameters (e.g., needs two arguments), Car breaks. This is **fragility of inheritance.**

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Fragile > Fragile_inheri.java

```java
// Parent class
class Vehicle {
    void drive(int speed) {
        System.out.println("Driving at " + speed);
    }
}

// Child class inherits from Vehicle
class Car extends Vehicle {
    @Override
    void drive(int speed) {
        super.drive(speed); // calls Vehicle's drive
        System.out.println("Car is moving");
    }
}

// Main method
public class Fragile_inheri {
    public static void main(String[] args) {
        Car car = new Car();
        car.drive(50); // works
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Fragile>javac Fragile_inheri.
java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Fragile>java Fragile_inheri
Driving at 50
Car is moving
```

Use of composition:

"Has-a" relationship

Real-Life Example

- A **Car has an Engine** → Composition
  The car *uses* the engine to run.

Engine can be replaced or upgraded without changing how the car is designed.

If inheritance were used:

- Car **is-a** Engine (wrong relationship)

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Composition > CountedListWrapper.java

```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

public class CountedListWrapper implements Iterable<String> {

    private final List<String> inner = new ArrayList<>();

    // Explicit implementation — no dependency on parent class internals
    public void add(String item) {
        inner.add(item);
    }

    public int getCount() {
        return inner.size();
    }

    // Optional extra wrapper methods
    public void addAll(Collection<String> items) {
        inner.addAll(items);
    }

    public String get(int index) {
        return inner.get(index);
    }

    @Override
    public Iterator<String> iterator() {
        return inner.iterator();
    }
}

```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Composition>javac Comp_Progra
m.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Composition>java Comp_Program

Count = 3
```

In Typescript:

: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Composition > TS comp_prg.ts > ...

```typescript
interface IList<T> {
    add(item: T): void;
}

// COMPOSITION FIX
class CountedListWrapper implements IList<string> {
    private inner: string[] = [];

    // Robust: No inheritance dependency
    add(item: string): void {
        this.inner.push(item);
        console.log(`Item added: ${item}, Current Count = ${this.inner.length}`);
    }
}

const list = new CountedListWrapper();
list.add("test");
list.add("Rithika");
list.add("Bank System");
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Composition>tsc comp_
prg.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Composition>node comp
_prg
Item added: test, Current Count = 1
Item added: Rithika, Current Count = 2
Item added: Bank System, Current Count = 3
```

Example: “Bank”

If BankAccount changes internally (maybe extra logic or new fields), Customer will not break easily as long as deposit/withdraw methods exist.

Customer

|-- has-a --> BankAccount

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Composition > J Composition.java

```java
// BankAccount class
class BankAccount {
    void deposit(double amount) {
        System.out.println("Deposited: " + amount);
    }

    void withdraw(double amount) {
        System.out.println("Withdrawn: " + amount);
    }
}


class Customer {
    private BankAccount account; // Customer has a BankAccount

    Customer(BankAccount account) {
        this.account = account;
    }

    void performTransactions() {
        account.deposit(1000);
        account.withdraw(500);
        System.out.println("Transaction completed successfully");
    }
}


public class Composition {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount(); // create account
        Customer customer = new Customer(acc); // attach to customer

        customer.performTransactions(); // using the account safely
    }
}

```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Composition>javac Composition
.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Composition>java Composition
Deposited: 1000.0
Withdrawn: 500.0
Transaction completed successfully
```

In Typescript:

EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Composition > TS Composition

```typescript
// BankAccount class
class BankAccount {
    deposit(amount: number) {
        console.log(`Deposited: ₹${amount}`);
    }

    withdraw(amount: number) {
        console.log(`Withdrawn: ₹${amount}`);
    }
}

// Customer class uses BankAccount (composition)
class Customer {
    private account: BankAccount; // Customer has a BankAccount

    constructor(account: BankAccount) {
        this.account = account;
    }

    performTransactions() {
        this.account.deposit(1000);
        this.account.withdraw(500);
        console.log("Transaction completed successfully");
    }
}

// Main execution
const account = new BankAccount();
const customer = new Customer(account);

customer.performTransactions();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Composition>tsc Compo
sition.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Composition>node Comp
osition
Deposited: ₹1000
Withdrawn: ₹500
Transaction completed successfully
```

Connascence?

How strongly two pieces of code depend on each other.

Unlike CBO (Coupling Between Objects), which only tells how many couplings, Connascence explains how strong the coupling is.

Type	Meaning	Coupling Strength
CoN of Name	Must share the same name	Weak / Good
CoN of Type	Must use same parameter types	Medium
CoN of Meaning / Algorithm	Must understand hidden rules or values	Very Strong / Bad

Connascence of Name:

- If we rename deposit() to addMoney(), the customer code breaks.
- This is Connascence of Name — both components must agree on the same name.

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Connascence > J CoN_Name.java

```java
class BankService {
    void deposit(double amount) {        // method name: deposit
        System.out.println("Deposited ₹" + amount);
    }
}

class Customer {
    private BankService bank = new BankService();

    void doDeposit() {
        bank.deposit(1000);              // must use same name 'deposit'
    }
}

public class CoN_Name {
    public static void main(String[] args) {
        new Customer().doDeposit();
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>javac CoN_Name.ja
va

D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>java CoN_Name
Deposited ?1000.0
```

In Typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Conna

```typescript
class BankService {
    deposit(amount: number) {
        console.log(`Deposited ₹${amount}`);
    }
}

class Customer {
    private bank = new BankService();

    doDeposit() {
        this.bank.deposit(1000);
    }
}

new Customer().doDeposit();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>tsc con_name.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>node con_name
Deposited ₹1000
```

Connascence of Type

- If transfer() changes to transfer(String amount), callers break.
  This is **Connascence of Type**.

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Connascence > CoN_Type.java

```java
class PaymentGateway {
    void transfer(double amount) { // expects double
        System.out.println("Transferred ₹" + amount);
    }
}

class ClientType {
    private PaymentGateway gateway = new PaymentGateway();

    void makeTransfer() {
        gateway.transfer(2500.50); // must pass correct type: double
    }
}

public class CoN_Type {
    public static void main(String[] args) {
        new ClientType().makeTransfer();
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java>cd Connascence

D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>javac CoN_Type.ja
va

D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>java CoN_Type
Transferred ?2500.5
```

In typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Connascence > TS con_type.ts >

```typescript
class PaymentGateway {
    transfer(amount: number) {
        console.log(`Transferred ₹${amount}`);
    }
}

class ClientType {
    private gateway = new PaymentGateway();

    makeTransfer() {
        this.gateway.transfer(2500.50); // must match type: number
    }
}

new ClientType().makeTransfer();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>tsc con_t
ype.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>node con_
type
Transferred ₹2500.5
```

Connascence of Meaning

Strong — BAD example using magic numbers & GOOD using enum

Bad Meaning:

- -1, 0, 1 have hidden meanings.
- Caller must **know secret meanings** (invisible fragile contract

In java

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Connascence > J CoN_Meaning_Bad.java

```java
class AccountServiceBad {
    // return codes:
    // -1 = insufficient funds, 0 = success, 1 = error
    int withdraw(double amount) {
        return -1;
    }
}

class ClientMeaningBad {
    private AccountServiceBad service = new AccountServiceBad();

    void attempt() {
        int code = service.withdraw(500);
        if (code == -1) {
            System.out.println("Insufficient funds (magic -1)");
        }
    }
}

public class CoN_Meaning_Bad {
    public static void main(String[] args) {
        new ClientMeaningBad().attempt();
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>javac CoN_Meaning
_Bad.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>java CoN_Meaning_
Bad
Insufficient funds (magic -1)
```

In typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Connascence > TS con_meaning_bad.ts > ...

```typescript
class AccountServiceBad {
    withdraw(amount: number): number {   // -1,0,1 magic numbers
        return -1;
    }
}

class ClientMeaningBad {
    private svc = new AccountServiceBad();

    attempt() {
        const result = this.svc.withdraw(500);
        if (result === -1) console.log("Insufficient funds (magic -1)");
    }
}

new ClientMeaningBad().attempt();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>tsc con_m
eaning_bad.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>node con_
meaning_bad
Insufficient funds (magic -1)
```

Good Meaaning: -Explicit meaning, No hidden protocol

In Java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Connascence > J CoN_Meaning_Good.java

```java
enum TxStatus { SUCCESS, INSUFFICIENT_FUNDS, ERROR }

class AccountServiceGood {
    TxStatus withdraw(double amount) {
        return TxStatus.INSUFFICIENT_FUNDS;
    }
}

class ClientMeaningGood {
    private AccountServiceGood service = new AccountServiceGood();

    void attempt() {
        TxStatus status = service.withdraw(500);
        if (status == TxStatus.INSUFFICIENT_FUNDS) {
            System.out.println("Insufficient funds (explicit enum)");
        }
    }
}

public class CoN_Meaning_Good {
    public static void main(String[] args) {
        new ClientMeaningGood().attempt();
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>javac CoN_Meaning
_Good.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Connascence>java CoN_Meaning_
Good
Insufficient funds (explicit enum)
```

In Typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Connascence > TS con_meaning

```typescript
type TxResult = "SUCCESS" | "INSUFFICIENT_FUNDS" | "ERROR";

class AccountServiceGood {
    withdraw(amount: number): TxResult {
        return "INSUFFICIENT_FUNDS";
    }
}

class ClientMeaningGood {
    private svc = new AccountServiceGood();

    attempt() {
        const result = this.svc.withdraw(500);
        if (result === "INSUFFICIENT_FUNDS")
            console.log("Insufficient funds (explicit)");
    }
}

new ClientMeaningGood().attempt();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>tsc con_m
eaning_good.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Connascence>node con_
meaning_good
Insufficient funds (explicit)
```

Examine the Depth of Inheritance:?

DIT = How many parent classes a class inherits from.

Example= “Bank”

Object

↑

Account (DIT = 0)

↑

SavingsAccount (DIT = 1)

↑

PremiumSavingsAccount (DIT = 2)

If we change Account.process() to remove super.process() or skip logic → all subclasses must update.

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Examine the dit > dit.java

```java
class Account {         // DIT = 0
    void process() {
        System.out.println("Account: Basic processing");
    }
}

class SavingsAccount extends Account {    // DIT = 1
    void process() {
        System.out.println("SavingsAccount: Add interest");
        super.process();
    }
}

class PremiumSavingsAccount extends SavingsAccount {   // DIT = 2
    void process() {
        System.out.println("PremiumSavings: Add rewards");
        super.process();
    }
}

public class dit {
    public static void main(String[] args) {
        PremiumSavingsAccount acc = new PremiumSavingsAccount();
        acc.process();
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java>cd examine the dit

D:\EI exercise\5Days-SoftwareArchitecture\Java\Examine the dit>javac dit.jav
a

D:\EI exercise\5Days-SoftwareArchitecture\Java\Examine the dit>java dit
PremiumSavings: Add rewards
SavingsAccount: Add interest
Account: Basic processing
```

In typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > examine dit > TS dit.t

```typescript
class Account {
  process() {
    console.log("Account: Basic processing");
  }
}

class SavingsAccount extends Account {
  process() {
    console.log("SavingsAccount: Add interest");
    super.process();
  }
}

class PremiumSavingsAccount extends SavingsAccount {
  process() {
    console.log("PremiumSavings: Add rewards");
    super.process();
  }
}

const acc = new PremiumSavingsAccount();
acc.process();
```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark>cd examine dit

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\examine dit>tsc dit.t
s

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\examine dit>node dit
PremiumSavings: Add rewards
SavingsAccount: Add interest
Account: Basic processing
```

Abstractness (A)?

Abstract contains interfaces or abstract classes

A= 1 → fully abstract (all interfaces/abstract classes)

A = 0 → fully concrete (all implemented classes)

Main Sequences?:

Graph with Abstractness (A) on Y-axis and Stability (I) on X-axis.

Distance from the Main Sequence (D)

$$D=|A+I-1|$$

D=0 (perfect), D=1(worst design)

Minimizing D ensures stable components are abstract, unstable components are concrete

Example:

BankInterface → stable & abstract → good (on main sequence)

CustomerManager → unstable & concrete → good

TransactionProcessor → stable & concrete → bad (below main sequence, fragile)

In java:

D: > EI exercise > 5Days-SoftwareArchitecture > Java > Main_seq > Main_seq.java

```java
// Abstract component (stable)
interface BankService {    // A = 1
    void processTransaction(double amount);
}

// Concrete component (unstable)
class CustomerManager implements BankService { // A = 0
    public void processTransaction(double amount) {
        System.out.println("Processing customer transaction: ₹" + amount);
    }
}

// Client code
public class Main_seq {
    public static void main(String[] args) {
        BankService service = new CustomerManager(); // depends on interface (good)
        service.processTransaction(500);
    }
}
```

```
D:\EI exercise\5Days-SoftwareArchitecture\Java>cd Main_seq

D:\EI exercise\5Days-SoftwareArchitecture\Java\Main_seq>javac Main_seq.java

D:\EI exercise\5Days-SoftwareArchitecture\Java\Main_seq>java Main_seq
Processing customer transaction: ?500.0
```

In Typescript:

D: > EI exercise > 5Days-SoftwareArchitecture > ts-benchmark > Main_seq > TS Main_seq.ts > ...

```typescript
// Abstract component
interface BankService {
    processTransaction(amount: number): void;
}

// Concrete component
class CustomerManager implements BankService {
    processTransaction(amount: number): void {
        console.log(`Processing customer transaction: ₹${amount}`);
    }
}

// Usage
const service: BankService = new CustomerManager();
service.processTransaction(500);

```

```
D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Main_seq>tsc Main_seq
.ts

D:\EI exercise\5Days-SoftwareArchitecture\ts-benchmark\Main_seq>node Main_se
q
Processing customer transaction: ₹500
```