OOD মূলনীতিসমূহ

...

অবজেক্ট ওরিয়েন্টেড ডিজাইন মূলনীতিসমূহ

OOP আর OOD এর মধ্যে পার্থক্য কি?

সেরা ১০টি অবজেক্ট ওরিয়েন্টেড ডিজাইন মূলনীতি

1. **SRP (Single Responsibility Principle)** – one class should do one thing and do it well
2. **Open Closed design principle** – open for extension, closed for modification
3. **LSP (Liskov Substitution Principle)** – Sub type must be substitutable for super type
4. **ISP (Interface Segregation Principle)** – Avoid monolithic interface, reduce pain on client side
5. **DIP (Dependency Inversion Principle)** – don’t ask, let framework give to you

Continue...

6. **DRY (Don't repeat yourself)** - avoids duplication in code
7. **Encapsulate what changes** – hides implementation detail, helps in maintenance
8. **Favor Composition over Inheritance** – Code reuse without cost of inflexibility
9. **Programming for Interface** – Helps in maintenance, improves flexibility
10. **Delegation principle** – Don't do all things by yourself, delegate it

1. Single Responsibility Principle (SRP)

1. Responsibility বলতে আসলে কি বুঝানো হচ্ছে?
2. Responsibility একটি হতে হবে বলতে কি বুঝানো হচ্ছে?
3. Responsibility কেন একটি হতে হবে?

> *“Gather together the things that change for the same reasons. Separate those things that change for different reasons.”*

```
public class Employee {
  public Money calculatePay();
  public void save();
  public String reportHours();
}
```

CEO কখন CFO, COO বা CTO কে চাকরীচ্যুত করবে?

```
class Book {

    public void getTitle() {
        return "A Great Book";
    }

    public String getAuthor() {
        return "John Doe";
    }

    public void turnPage() {
        // pointer to next page
    }

    public void printCurrentPage() {
        System.Print(“current page content");
    }
}
```

```
class Book {

    public String getTitle() {
        return "A Great Book";
    }

    public String getAuthor() {
        return "John Doe";
    }

    public void turnPage() {
        // pointer to next page
    }

    public String getCurrentPage() {
        return "current page content";
    }

}

interface Printer {

    public void printPage(string page);
}
```

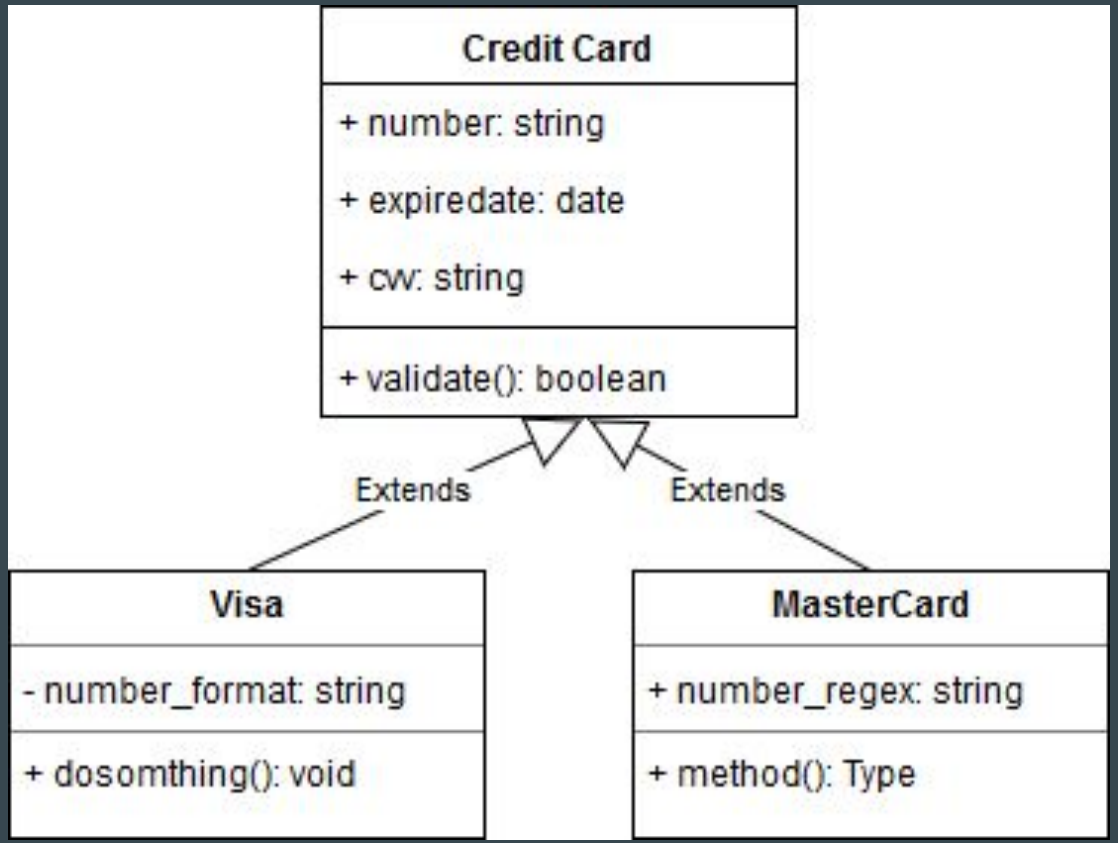
```
class PlainTextPrinter implements Printer {

    public void printPage(string page) {
        System.Print(page);
    }

}

class HtmlPrinter implements Printer {

    public void printPage(string page) {
        System.Print(“'<div style=’single-page’>”
+ page + “</div>”);
    }

}
```

2. Open-Closed Principle (OCP)

1. কোন কোড লেখা হয়ে গেলে যেন সেটা কেউ পরিবর্তন না করে।
2. কিন্তু পরিবর্তনের প্রয়োজন হলে যেন সেটা করার একটা পথ থাকে।
3. মূল ক্লাস অপরিবর্তিত থাকবে কিন্তু বাইরে থেকে Inheritance বা Composition এর মাধ্যমে প্রয়োজনীয় পরিবর্তন আনতে হবে।

“A class should be closed for modification and open for extension.”

2. Open-Closed Principle (OCP)

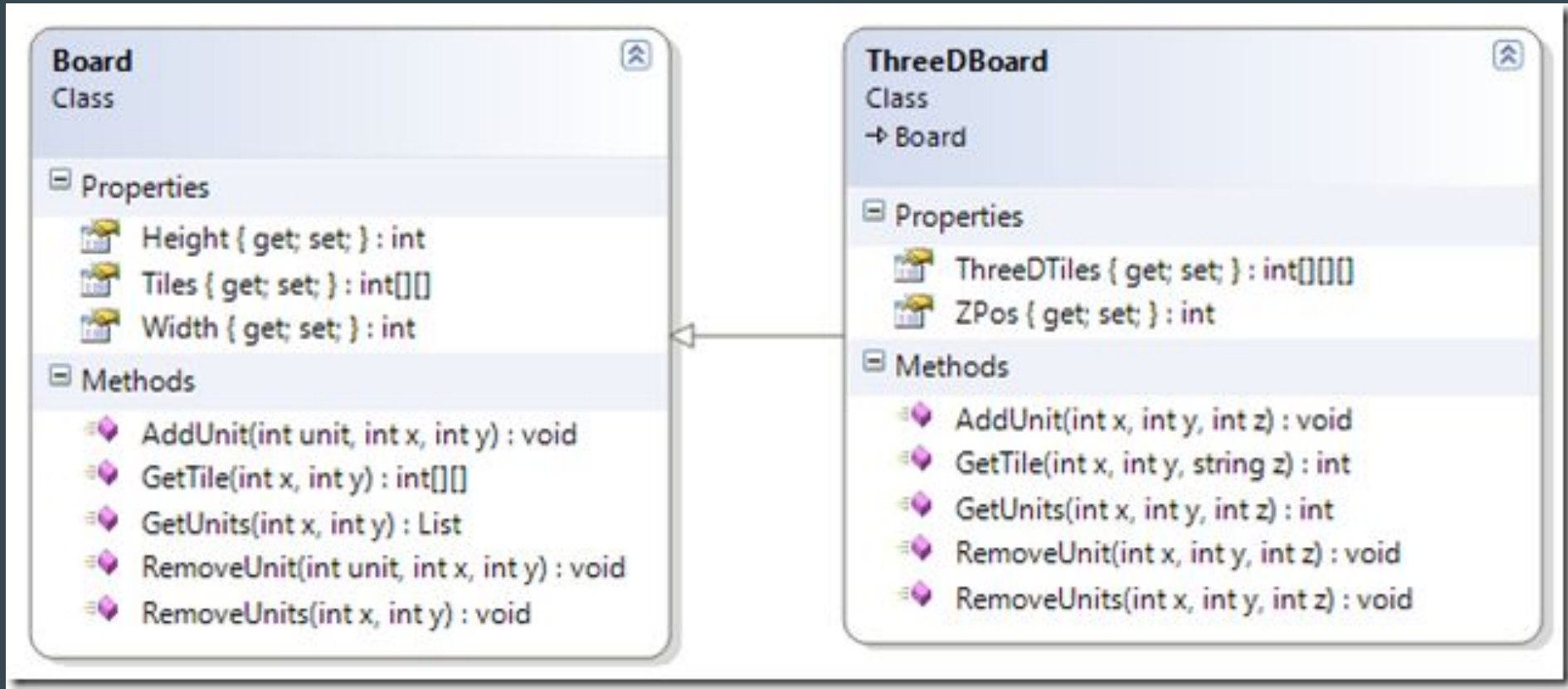


3. Liskov Substitution Principle (LSP)

1. যদি একটি ক্লাস থেকে অন্য একটি ক্লাসকে Inheritance করা হয় তাহলে এমন হতে হবে যে Parent Class ও সকল Child Class একইভাবে ব্যাবহার করা যাবে।
2. Parent Class ও সকল Child Class ব্যাবহারের ক্ষেত্রে কোন বিশেষ নিয়ম পালনের বা চিন্তা করার প্রয়োজন পরবে না।
3. “Is a” Relation দিয়ে আমরা অনেক সময় এটা চেক করতে পারি তবে মনে রাখতে হবে, বাস্তব দুনিয়াতে “Is a” Relation যেভাবে কাজ করবে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর দুনিয়াতে সেটা একইভাবে কাজ করবে এটা ভাবা ভুল হবে। কাজেই কোডে ব্যাবহার করার সময় আমরা কোন কিছু মাথায় রেখে কোড করছি নাকি একই ভাবে কোন চিন্তা ভাবনা ছাড়াই Parent Class ও সকল Child Class কে ব্যাবহার করতে পারছি এটার দিকে বিশেষ নজর রাখতে হবে।

3. Liskov Substitution Principle (LSP)

3. Liskov Substitution Principle (LSP)



4. Interface Segregation Principle (ISP)

1. বড় Interface এর পরিবর্তে ছোট ছোট Interface তৈরি করতে হবে।
2. Interface যেন আমাদের ক্লাসে নতুন বৈশিষ্ট্য যোগ করার কাজে ব্যাবহার করা যায়।
3. Interface একাধিক ক্লাসে ব্যাবহার করা সম্ভব হয়।

“No client should be forced to depend on methods it does not use”

4. Interface Segregation Principle (ISP)

Big Fat Interface

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ISPDemoConsole
{
    public interface IPrintTasks
    {
        bool PrintContent(string content);
        bool ScanContent(string content);
        bool FaxContent(string content);
        bool PhotoCopyContent(string content);
        bool PrintDuplexContent(string content);
    }
}
```

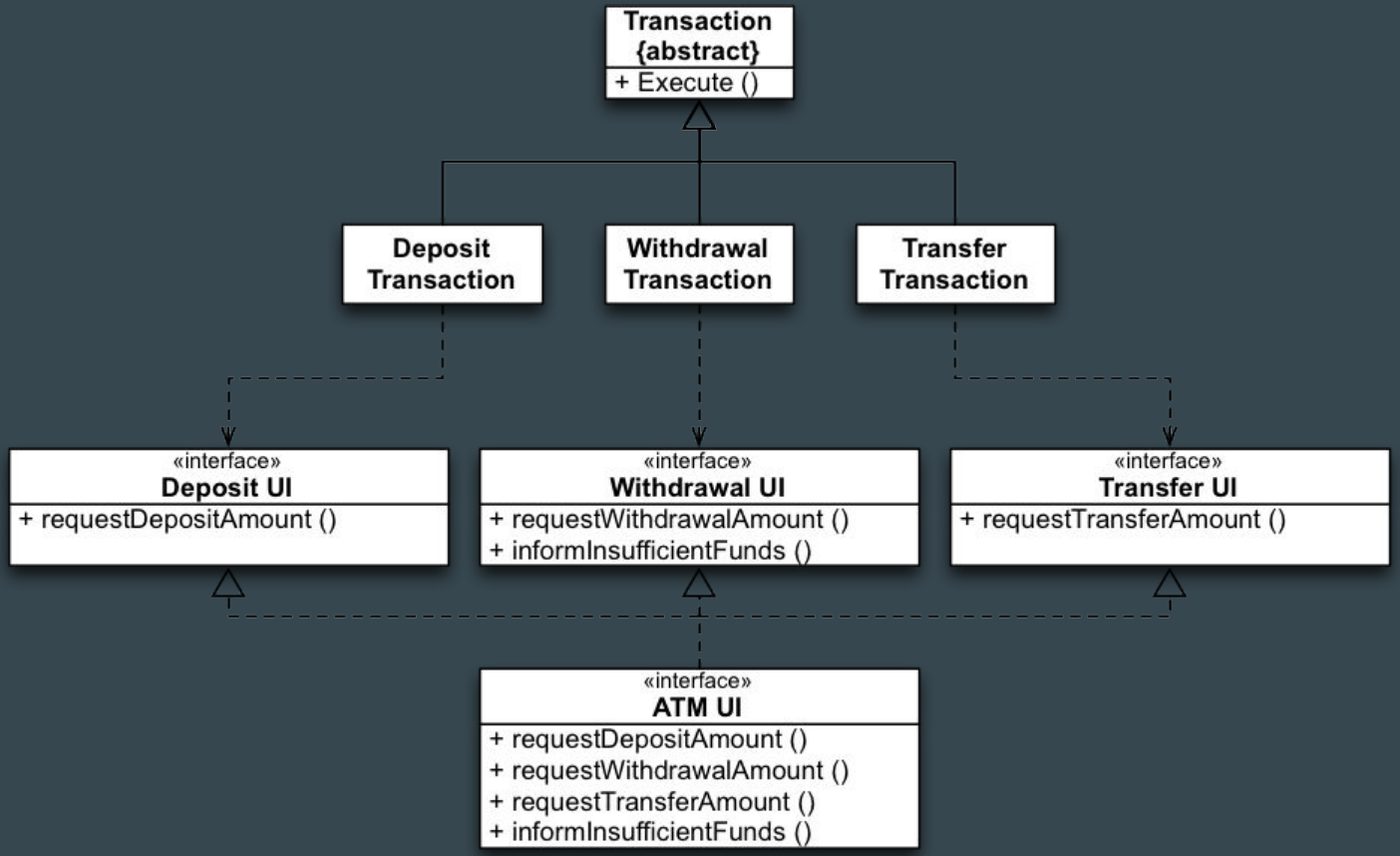
Smaller Interfaces

```
interface IPrintScanContent
{
    bool PrintContent(string content);
    bool ScanContent(string content);
    bool PhotoCopyContent(string content);
}
```

```
interface IFaxContent
{
    bool FaxContent(string content);
}
```

```
interface IPrintDuplex
{
    bool PrintDuplexContent(string content);
}
```

4. Interface Segregation Principle (ISP)



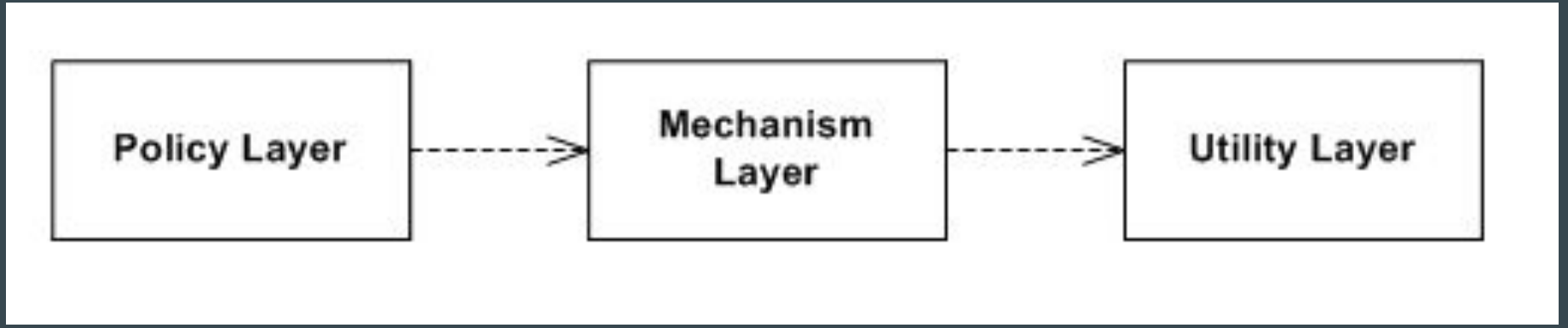
5. Dependency Inversion Principle (DIP)

1. কোডের মধ্যে Dependency দূর করার জন্য এটি ব্যাবহার করা হয়।
2. এমনভাবে আমরা ২ টি মডিউলকে ডিজাইন করবো যেন কেউ কারো উপর সরাসরি নির্ভর না করে বরং Interface এর উপর নির্ভর করে।

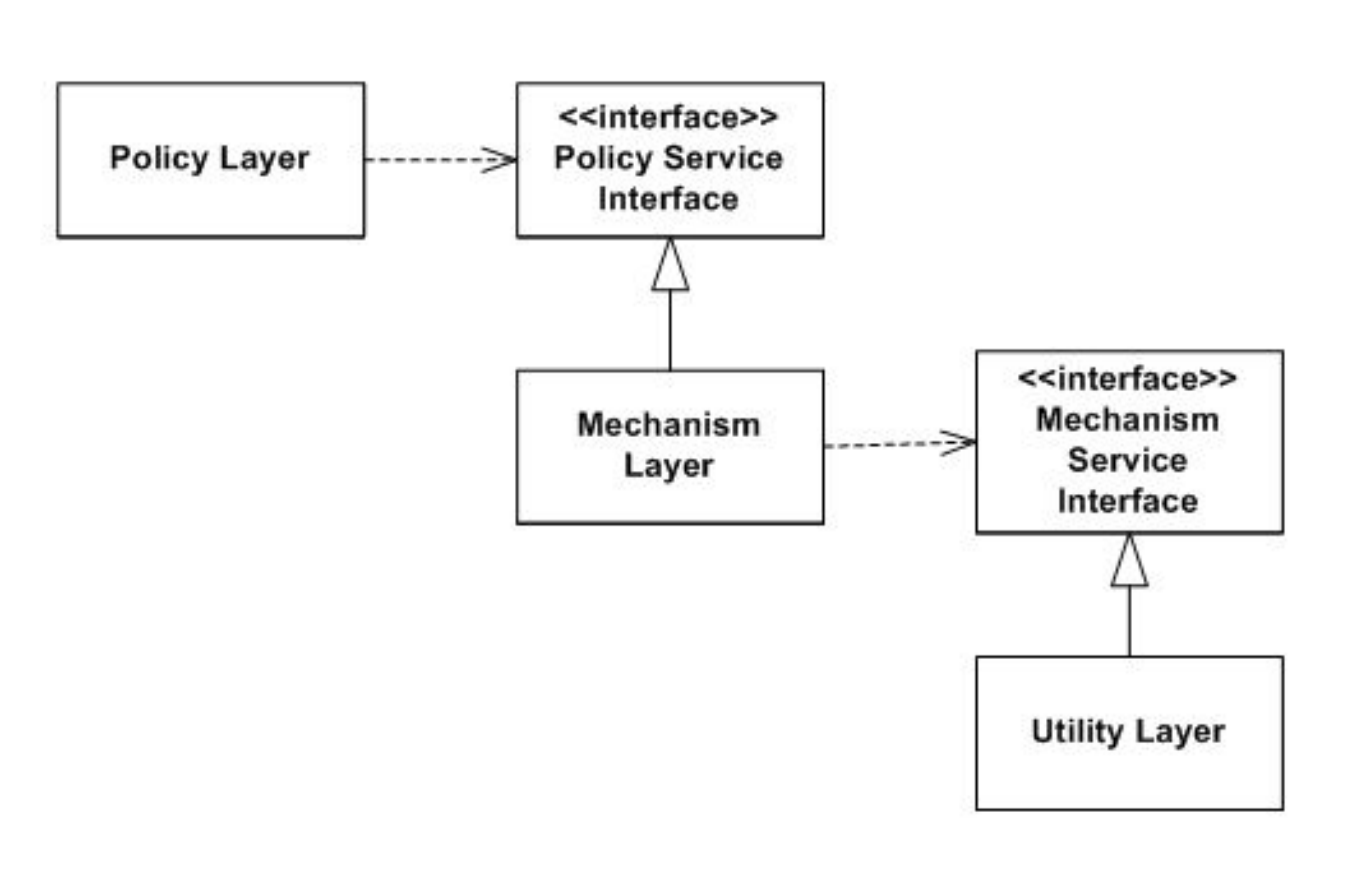
“High-level modules should not depend on low-level modules. Both should depend on abstractions”

“Abstractions should not depend on details. Details should depend on abstractions.”

5. Dependency Inversion Principle (DIP)



5. Dependency Inversion Principle (DIP)



7. Encapsulate what changes

```
if (pet.type() == dog) {
  pet.bark();
} else if (pet.type() == cat) {
  pet.meow();
} else if (pet.type() == duck) {
  pet.quack()
}
```

I may reasonably expect that this piece of code will change in the future. I can "encapsulate" it simply by defining a new procedure:

```
void speak(pet) {
  if (pet.type() == dog) {
    pet.bark();
  } else if (pet.type() == cat) {
    pet.meow();
  } else if (pet.type() == duck) {
    pet.quack()
  }
}
```