what is Design pattern?

is a reusable solution to a commonly occuring problem in software design.

Creational Design	Structural Design	Behavioral Design
→ object creation mechanism provide way to create object without specifying their class	→ How to assemble objects and class into large structure while keeping the structure flexible and efficent	→ concerned with the interaction and communication objects defining the How and Responsibility
① Singleton	① Adapter	① State
② Factory	② Composite	② Strategy
③ Abstract Factory	③ Proxy	③ Chain of Responsibility
④ Prototype	④ Decorator	④ observer
⑤ Builder	⑤ Template	⑤ visitor
	⑥ Facade	⑥ command
	⑦ Flyweight	⑦ Memento
		⑧ Iterator

codesmell :- refers to certain patterns / characteristic in code that indicate potential design and Implementation issue

Refactoring :-

is the process of restructing extisting code for impoving design, readaptibility, maintainability without changing its external behaviour

1. Inappropiate Naming
2. Dead code
3. comment
4. primitive obsession
5. switch statement
6. Large class
7. Lazy class
8. Long parameter List
9. Long method
10. Odd ball solution
11. Feature Envy
12. Refused Bequest
13. Blacksheep
14. Speculative generality
15. Duplicate code

violates Single Responsibility principle

1. Large class
2. Long method
3. Long parameter List
4. primitive obsession

Feature Envy (violates Dependency Inversion principle)

occurs when a method excessively uses (data / behaviour / methods / properties) from another class instead of It own

```
class customer {
    string Name;
    Address address;

public string getaddressInfo()
{   string details = "Address:"
    + address.getStreet()
    + address.getcity()
    + address.getcountry();
return details }
```

calling 4[2,2,2]

```
class Address {
  private string street;
      "      "    city;
      "      "    country;

public string getStreet()
  { return street }

public string getcity()
  {                 }

public string getcountry()
  {                 }
```

solve:-

```
getaddness info ()
{
  details = "Address:
  + address.getAll()
}
```

```
string
public getAll() {
  return getstreet()
  +getcity()
  +getcountry()
}
```

— o —

oddball solution

refers to non-standard and unconventional way of solving a problem

```
public class Math {
    public int add (int a, int b) {
        return a - (-b);
    }
}
```

Refused Bequest

occuns when subclass inherits from superclass but doesn't fulfill on use the contracts on expectations of inherited methods.

Liskov subsitiution Principle

delagation :- is a design principle where an objects forwards its responsibility to another object.

Instead of ~~inheritance~~ inheriting behaviour from super class, the delagating objects holds refenence to another object, and call its method to perform task.

(1) promotes loose coupling :- as delagating object donot need to know how delageted objworks

(2) Enable runtime flexibility :- allows diffenent behavior by swapping delagating objects

(3) encourage better separation of concenns :- each object foucus on its own responsibility

Delagting class

```
class printer {
private printservice p;
void print(string doc)
{ p.print(doc); }
```

forward

Delagated class

```
class printservice {
void print(string doc)
{ sout(doc); }
}
```

object composition

is a design principle where a class is composed of one or more objects of other class (instance variable)

→ allow building complex objects by combining smaller components

Advantage

1. reuse of code small components
2. encourage modular approach and easy maintain the code
3. more flexible than inheritance

disadvantage

→ increase complexity

→ increase memory usage

```
class Engine {
    void start();
}

class wheels {
    void Rolls();
}

class Body {
    void show(); }
```

class car {

```
private Engine e;
private wheels w;
private Body B;

void drive() {
    e.start()
    w.Rolls()
    B.show() }
```

(3) Object Inheritance

refers to where a class (subclass) inherits properties and behaviour from another class (superclass)

1. code reusebility
2. allows polymorphism

disadvantage

1. tight coupling between classes

```
class Bird {
    void fly() {
    }
}
```

```
class Dove
    extends Bird {
    void fly() {
    }
}
```

four essential element of Design pattern

1. Pattern Name
2. Problem
3. solution
4. consequence
   (result a tradeoff)

How to Describe a pattern

1. Pattern name and classification
2. Intent
3. motivation
4. Applicability
5. structure
6. participants
7. consequence
8. Implementation
9. sample code
10. known use
11. Related pattern

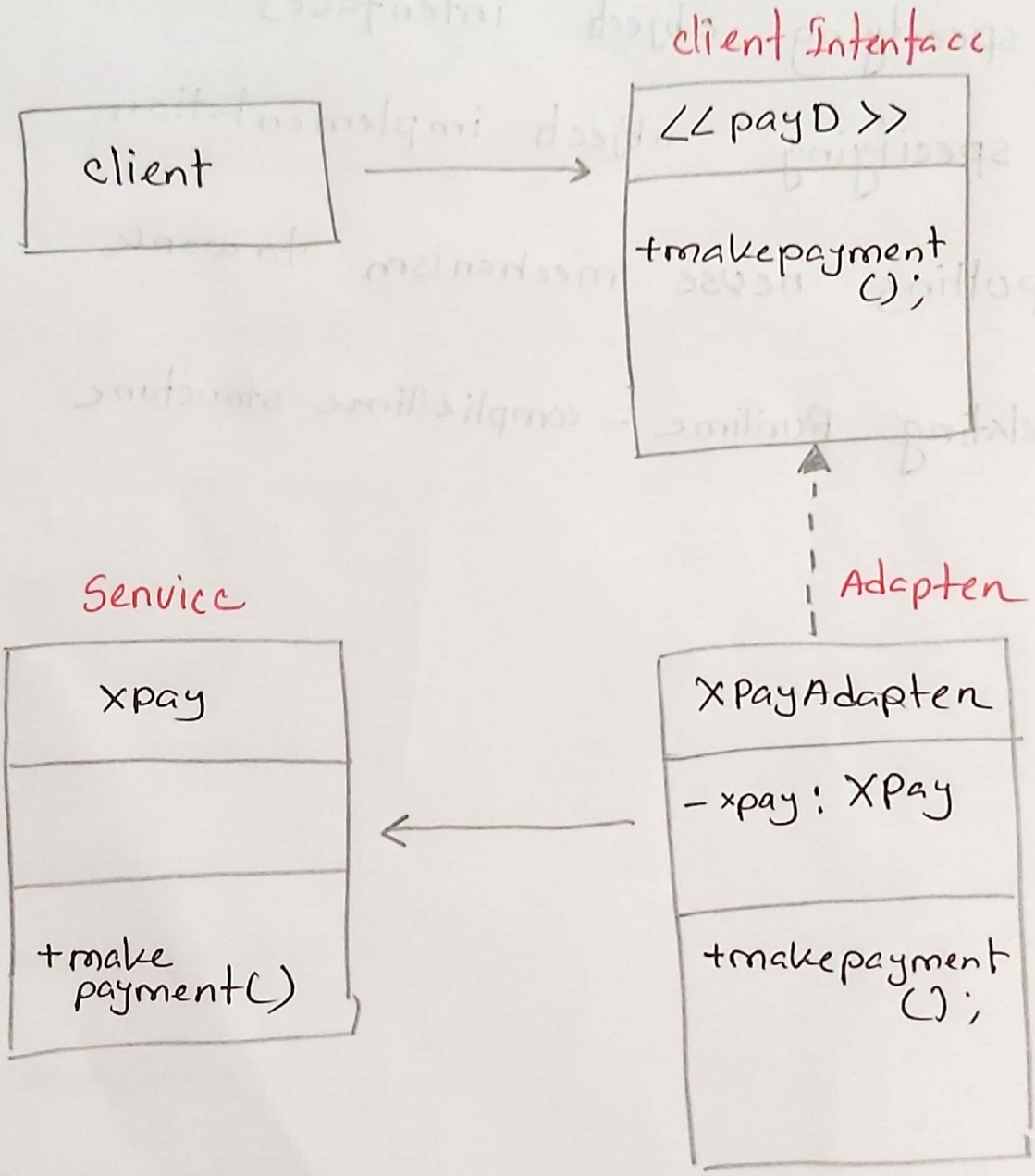
How design pattern solve design problem?

1. Finding appropriate objects
2. Determining object Granularity
3. specifying objects interfaces
4. specifying objects implementation
5. putting reuse mechanism to work
6. Relating Runtime - compile Time structure

(16) question 6c

Adapter design pattern

is a structural pattern that allows objects with incompatible interface to collaborate



① interface PayD {
void makepayment()
}

② class xpayAdapter
implements PayD {
private xpay xpay;
xpayAdapte (Xpay xpay)
{ this.xpay = xpay }

void makepayment()
{ xpay. makepayment() }

③ class Xpay {
void makepayment()
{ ... }
}

④ main {
Xpay p = new Xpay;
PayD pay = new
xpayAdapter
(P);
pay. makepayment();
}