Here is **every** significant PYQ pattern for OOPs, categorized for your final revision.

Category 1: Polymorphism & Virtual Functions (The #1 Topic)

Q1. (2023, 2021) Output of the code?

```
C++

class Base {
public:
   virtual void print() { cout << "Base"; }
};
class Derived : public Base {
public:
   void print() { cout << "Derived"; }
};
int main() {
   Base *b = new Derived();
   b->print();
}
```

- **Answer:** Derived
- **Logic:** The function is virtual, so the decision is made at **Runtime** based on the object (Derived).

Q2. (2022) Output of the code?

```
C++

class Base {
public:
   void print() { cout << "Base"; } // NOT VIRTUAL
};
class Derived : public Base {
public:
   void print() { cout << "Derived"; }
};
int main() {
   Base *b = new Derived();
```

```
    b->print();
}
```

- **Answer:** Base
- **Logic:** The function is **NOT** virtual. The decision is made at **Compile-time** based on the pointer type (Base*).

Q3. (2020) Pure Virtual Function Syntax

Which statement declares a pure virtual function?

- A) virtual void func() {}
- B) virtual void func() = 0;
- C) virtual void func() { return 0; }
- D) void func() = 0;
- **Answer: B**

Q4. (2019) Abstract Class

If a class has one pure virtual function, can we create an instance of it?

- **Answer: No**. It becomes an Abstract Class.

Q5. (2021) Virtual Destructor

What happens if the Base class destructor is NOT virtual?

- **Answer:** Only the Base destructor is called; the Derived destructor is skipped, causing a **memory leak**.

Category 2: Constructors & Destructors

Q6. (2024, 2018) Order of Execution

```
C++
class Parent {
public:
    Parent() { cout << "P"; }
    ~Parent() { cout << "p"; }
};
```

```
class Child : public Parent {
public:
   Child() { cout << "C"; }
   ~Child() { cout << "c"; }
};
int main() { Child c; }
```

- **Answer:** PCcp
- **Logic:** Constructor: Parent $\to$ Child. Destructor: Child $\to$ Parent.

Q7. (2022) Copy Constructor

Which syntax is correct for a Copy Constructor?

- A) `Class(Class obj)`
- B) `Class(const Class &obj)`
- C) `Class(Class *obj)`
- D) `Class(const Class obj)`
- **Answer: B** (Must pass by reference `&` to avoid infinite recursion).

Q8. (2020) Explicit Constructor

What does the keyword explicit do?

- **Answer:** It prevents **implicit type conversion** by the compiler.

Q9. (2019) Constructor Overloading

Can constructors be overloaded?

- **Answer: Yes** (You can have Default, Parameterized, and Copy constructors in the same class).

Category 3: Static Members & `this` Pointer

Q10. (2023) Static Variable Output

```
C++

class Test {
   static int i;
```

```
    int j;
};
int Test::i;
int main() {
    cout << sizeof(Test);
}
```

- **Answer:** 4 (Assuming int is 4 bytes).
- **Logic: Static variables are NOT stored inside the object.** They are stored separately. The object only contains j.

Q11. (2021) Static Function Limitations

Which members can a static member function access?

- **Answer:** Only **Static** data members and other **Static** functions. It cannot access non-static members.

Q12. (2022) this Pointer

Does a static member function have a this pointer?

- **Answer: No**.

Q13. (2018) Initialization of Static

Where must static member variables be defined?

- **Answer: Outside the class definition** (Global scope). e.g., int ClassName::variableName = 0;.

Category 4: Inheritance & Access Specifiers

Q14. (2020) The Diamond Problem

Which inheritance type causes ambiguity where one class inherits from two classes that have a common base?

- **Answer: Multiple Inheritance** (specifically the "Diamond" pattern).

Q15. (2022) Solving Ambiguity

How do we resolve the Diamond Problem?

- **Answer:** Using **Virtual Inheritance** (`class B : virtual public A`).

Q16. (2019) Access Rights

If a member is protected, who can access it?

- **Answer:** The class itself, **friend functions**, and **derived classes**. (Outside world cannot access it).

Q17. (2021) Default Inheritance

If you write `class B : A { ... };`, what is the default access mode?

- **Answer: Private** (In Classes, default is Private. In Structs, default is Public).

Category 5: Operator Overloading & Friends

Q18. (2023, 2019) Forbidden Operators

Which operator cannot be overloaded?

- **Answer:** `::` (Scope Resolution), `.` (Dot), `.*` (Pointer to member), `?:` (Ternary), `sizeof`.

Q19. (2020) Friend Function Property

Is friendship transitive? (If A is friend of B, and B is friend of C, is A friend of C?)

- **Answer: No**.

Q20. (2018) Unary Operator Overloading

How many arguments does a member function take to overload a unary operator (like ++)?

- **Answer: Zero** (0).
- *Note:* If it were a friend function, it would take **One** (1).

Category 6: Miscellaneous & Tricky Questions

Q21. (2022) Empty Class Size

What is the output of cout << sizeof(EmptyClass);?

- **Answer: 1** (Byte).

Q22. (2021) Function Overloading

Function overloading is determined by:

- A) Return type
- B) Number and Type of Arguments
- C) Both
- **Answer: B** (Return type alone is NOT sufficient for overloading).

Q23. (2023) Inline Functions

An inline function is a request to the compiler to:

- **Answer: Replace the function call with the actual code** (to reduce function call overhead).

Q24. (2019) Exception Handling

What catches all exceptions?

- **Answer:** `catch(...)`

Q25. (2020) Reference Variable

Output?

```C++
int x = 10;
int &y = x;
y = 20;
cout << x;
```

- **Answer:** `20`
- **Logic:** `y` is a reference (alias) to `x`. Changing `y` changes `x`.

Final Advice for Day After Tomorrow

You have now seen **every** major question type.

- **Don't memorize code snippets letter-by-letter.** Memorize the **Logic** (Virtual = Runtime, Static = Shared, Private = Hidden).
- **Watch out for `sizeof` questions**—they are easy marks.