📘 **COMPLETE NOTES**

INNER CLASSES + ANONYMOUS CLASSES + LAMBDA EXPRESSIONS

1️⃣ NUMBER OF .class FILES CREATED

✅ Rule:

Number of compiled .class files = Number of classes (including inner & anonymous classes)

Example:

```
class Person {
   class Phone { }
}
```

After compilation:

Person.class

Person$Phone.class

If anonymous class is present:

Person$1.class

2️⃣ INNER CLASS (NON-STATIC INNER CLASS)

✅ **Definition**

A class inside another class.

```
class Person {
   int age;
   String name;

   class Phone {
      String company;
      String model;
   }
}
```

🔹 **Why Inner Class?**

Used when:

- One class is strongly dependent on another
- Logical grouping
- Improve encapsulation
- Used only by outer class

Example:
Phone belongs to Person → Logical relationship

🔹 **How to Create Object of Inner Class**

Non-static inner class needs outer class object.

Person p = new Person();

Person.Phone ph = p.new Phone();

⚠ Important:

Inner class object cannot exist without outer class object.

🔹 **Access Rules**

Inner class CAN access:

- Private members of outer class

Outer class CANNOT directly access:

- Inner class members without object

3️⃣ **STATIC INNER CLASS (STATIC NESTED CLASS)**

```
class Person {
   static class Phone {
      void details() {
         System.out.println("Phone details");
      }
   }
}
```

🔹 **Why Static Inner Class?**

Used when:

- Inner class does not depend on outer object
- Logical grouping
- Utility/helper class

🔹 **How to Create Object**

Person.Phone obj = new Person.Phone();

obj.details();

⚠ No need of outer class object.

🔹 **Important Rule**

Static inner class:

- Can access only static members of outer class
- Cannot access non-static members directly

4️⃣ **ANONYMOUS INNER CLASS**

🔹 **What is it?**

A class without name.

Used when:

- Implementation required only once
- No reuse needed

Example:

```
abstract class A {
   abstract void show();
}A obj = new A() {
   void show() {
      System.out.println("Hello");
   }
};
```

🔹 Why Use Anonymous Class?

Instead of:

```
class B extends A {
   void show() {
      System.out.println("Hello");
   }
}
```

If implementation needed only once → use anonymous.

🔹 Important Points

✓ Cannot reuse
✓ No constructor name
✓ One-time usage
✓ Creates separate .class file (A$1.class)

5️⃣ ABSTRACT CLASS VS ANONYMOUS CLASS

Your Question:

If abstract class exists, why extend and create class for single implementation?

Answer:

If implementation needed:

- Many times → create separate class
- One time only → use anonymous class

6️⃣ FUNCTIONAL INTERFACE

🔹 Definition

An interface with exactly ONE abstract method.

Example:

```
interface A {
   void show();
}
```

Optional annotation:

```
@FunctionalInterface
interface A {
   void show();
}
```

🔹 **Why Introduced?**

To support:

- Lambda expressions
- Functional programming

7️⃣ LAMBDA EXPRESSION

🔥 **Why Lambda Came?**

Before Java 8:

```
Runnable r = new Runnable() {
   public void run() {
      System.out.println("Running");
   }
};
```

Too much boilerplate code.

Java 8 introduced Lambda.

🔹 **Definition**

Lambda Expression = Short form of anonymous class for functional interfaces.

🔹 **Syntax**

```
(parameters) -> { body }
```

Example:

```
Runnable r = () -> {
   System.out.println("Running");
};
```

🔹 **With Parameter**

```
interface Add {
   int sum(int a, int b);
}

Add obj = (a, b) -> a + b;
```

8️⃣ **INTERNAL WORKING**

Anonymous Class:

Creates new .class file:

Outer$1.class

Lambda:

Does NOT create separate class file like anonymous.

It uses:

invokedynamic

bytecode instruction.

More efficient.

9️⃣ **DIFFERENCE: ANONYMOUS VS LAMBDA**

Feature	**Anonymous Class**	**Lambda**
Syntax	Long	Short
Class file	Yes	No extra visible class
this keyword	Refers to inner class	Refers to outer class
Can define new methods	Yes	No

Feature	Anonymous Class	Lambda
Use for	Any interface	Only functional interface

🔟 IMPORTANT CONCEPTS

✓ this keyword

Anonymous:
this → refers to anonymous class

Lambda:
this → refers to outer class

✓ Lambda cannot:

- Extend class
- Create object independently
- Work with multiple abstract methods
- Have instance variables

1️⃣1️⃣ TYPES OF INNER CLASSES

Java has 4 types:

1. Member Inner Class
2. Static Nested Class
3. Local Inner Class
4. Anonymous Inner Class

1️⃣2️⃣ LOCAL INNER CLASS

Defined inside method.

```
void display() {
   class Local {
      void show() {}
   }
}
```

Accessible only inside method.

1️⃣3️⃣ METHOD REFERENCE (Related to Lambda)

Instead of:

(a) -> System.out.println(a)

Use:

System.out::println

1️⃣4️⃣ WHEN TO USE WHAT?

Situation	Use
Strong outer relationship	Inner class
No outer dependency	Static inner
One-time implementation	Anonymous
Single abstract method	Lambda
Multiple methods	Normal class

1️⃣5️⃣ YOUR IMPORTANT DOUBT CLEARLY

You asked:

Why extend abstract class when we can use anonymous?

Answer:

If implementation:

- Reusable → Create separate class
- One-time → Anonymous
- Functional interface → Lambda

1️⃣6️⃣ INTERVIEW TRAPS

1. Can lambda replace abstract class? ❌ No
2. Can lambda implement multiple methods? ❌ No
3. Does lambda create .class file? ❌ Not like anonymous
4. Can lambda have constructor? ❌ No

5. Can lambda access non-final variable? ❌ Only effectively final

---

1️⃣7️⃣ COMPLETE FLOW UNDERSTANDING

Normal Class →
Inner Class →
Static Nested Class →
Anonymous Class →
Functional Interface →
Lambda Expression

Evolution of simplification in Java.

🚀 FINAL SUMMARY

Inner Classes → Code organization
Anonymous Class → One-time implementation
Functional Interface → Single abstract method
Lambda → Short form of anonymous class