Lesson-5 – Class Notes

Default and Protected have package level access – both are same if your class has no Inheritance
If you have Inheritance relationship, even Protected can access other package too

Inheritance - IS - A relationship / Generalization.

Super class / Parent class - Top class - More General (Common)

Child class / Derived class (More specific, It can access everything from the parent except private fields and methods

1. How to create Sub class
2. Use of super keyword
3. Declare object using Super type and assigning sub type to super type
4. Need of overriding
5. Polymorphism - Dynamic binding
6. Calling set Bonus from Employee object error
7. instance of
8. Rules about Constructor in Inheritance

If you have no constructors in both Parent and Child – Default constructor is automatic – No Compilation Error

My Parent class have Default Constructor and my child class has no constructor – fine, No compilation error – Automatic

If you have no default constructor in your parent, you have emplicit constructor in your parent, must make use of any one parent constructor in your child class.

Employee Manager Example
Right Usage - Inheritance

1. You should have proper IS-A relationship, Example Manager is a Employee
2. Your inheritance must support Liskov Substitution principle(LSP),
   If I need a type of Manager instance I can able substitute Employee instance

```
main(){
      employee arraylist collections, Linked list manager collections
      print(employee)
      print(manager);

// API

}
```

```
public void print(List[] ea){
  // Do something
}
```

Java Does not support Multiple inheritance

Legal - Class B extends A
illegal - Class C extends A,B --> (Does not support multiple inheritance)
TO overcome the Multiple inheritance issue java supports to use multiple interface.
Java Does not support Multiple inheritance

Legal - Class B extends A
illegal - Class C extends A,B --> (Does not support multiple inheritance)
TO overcome the Multiple inheritance issue java supports to use multiple interface.

About Number Type – Arithmetic operations

The Number class in Java is an abstract superclass for numeric wrapper classes like Integer, Double, Float, Long, Short, and Byte. It provides methods to convert its instances into different primitive types. However, it does not directly support arithmetic operations because it serves as a generalized abstraction.

```
Number x = 20;
Number y = 20;
//Number z = x + y; // Cause compilation error
```

Right Usage

```
Number x = 20;
Number y = 20;
Double z1 = x.doubleValue() + y.doubleValue();
Number z2 = x.doubleValue() + y.doubleValue();
int sum = x.intValue() + y.intValue();
```

The Number class primarily provides methods for type conversion and polymorphic behavior. While you cannot directly perform arithmetic or other operations on Number, you can use type-specific methods (like intValue(), doubleValue()) to work with the underlying value.

Key differences between **method overloading** and **method overriding** in Java:

Aspect	**Overloading**	**Overriding**
Definition	Defining multiple methods with the same name but different parameter lists (type, number, or order).	Redefining a method in a subclass that already exists in the parent class with the same signature.
Relationship	Happens within the same class.	Happens between a parent class and a subclass.
Parameters	Requires a difference in the number or type of parameters.	The method signature (name, parameters, return type) must remain the same.
Return Type	Can have different return types if parameters differ.	The return type must be the same
Access Modifier	No restrictions on access modifiers.	The access level can be higher visibility than the parent method or same level of access modifier.
Binding (Runtime/Compile-time)	Resolved at compile time (static binding).	Resolved at runtime (dynamic binding).
Annotations	Does not require annotations.	Typically marked with @Override to ensure correctness.
Use Case	Used to increase the readability of the code by allowing the same method name for different operations.	Used to provide specific behavior in a subclass while keeping the method signature consistent.

Lesson 5 – Day – 2

To achieve Polymorphism to perform OO approach, you need a common type / Generalization

- Regular Inheritance / Abstract class
- Interfaces

Circle, Square, Triangle – All three has no relationship and no common type(Then we can use Object type)

Create an array of all those three collection

```
Object arr[] = new Object{new Circle(34.5),
                          new Triangle(5.6.7.8),
                          new Square(5)};
for(Object ob : arr)
```

```
 {
    Ob.ComputerArea();  // You cannot access
   If(ob instance of Circle)
         Circle c = (Circle) ob;
}
```

Demo Purpose – Abstract class – Book – Subclass EBook and PaperBook

Interface

Pre-Java-8 Interface

- ➔ Pre-Java-8 – All methods are unimplemented, by default abstract methods
- ➔ By default all the fields are public static final
- ➔ By default all the methods public abstract → Unimplemented
- ➔ You cannot create object for an interface
- ➔ You cannot create constructor inside an interface
- ➔ You sub class can implements Interface
- ➔ One class can implement multiple interface
- ➔ One interface can inherit from other interface using extends keyword. May inherit one or multiple interface

Drawbacks from Pre-Java-8

If you add some methods in the interface it affects your implementation. TO overcome this drawback Java 8 introduces two implemented methods

1. Default
2. Static

You can able to override the Default implementation inside the class or you can go with interface default implementation.

JDK 9

You can include private methods inside interface

Here's a tabulated comparison between Java Abstract Class and Interface:

Feature	**Abstract Class**	**Interface**
Definition	A class that can have both abstract methods (without implementation) and concrete methods (with implementation).	A collection of abstract methods (prior to Java 8). Starting with Java 8, it can have default and static methods.
Multiple Inheritance	Does not support multiple inheritance (a class can extend only one abstract class).	Supports multiple inheritance (a class can implement multiple interfaces).

Feature	Abstract Class	Interface
Method Types	Can have abstract, concrete, static, and final methods.	Can have abstract methods, default methods, static methods, and private methods (since Java 9).
Fields	Can have instance variables (state) and static fields.	Can only have public static final variables (constants).
Access Modifiers	Methods and fields can have any access modifier (public, protected, private).	Methods are public by default. Variables are public static final by default.
Constructors	Can have constructors.	Cannot have constructors.
Purpose	Used for creating a base class with shared behavior that can be inherited.	Used for defining a contract that implementing classes must adhere to.
Keyword to use	Use extends keyword to inherit.	Use implements keyword to implement.
When to Use	Use when you need a shared base class with some common implementation.	Use when you need to define a contract for classes without sharing implementation.

Reflection Library

Retrieving a Class type is a fundamental part of Java Reflection. It allows you to examine or modify the runtime behavior of applications. For example, you can dynamically create objects, access fields, invoke methods, and much more.

Class.forName() is useful for dynamic class loading, where the class name is not known until runtime. This is particularly useful in frameworks and libraries that need to load user-defined classes or configurations.

Functional Interface – An interface has only one unimplemented method. This can be implemented Lambdas. Example: Comparable

Marker Interface – Interface has no methods is called Mark interface. (No methods which useful to do cloning, data persistency). Example - API: Serializable

API example – Comparable Interface – Functional Interface

```
@FunctionalInterface
Interface test{
 void display();
}
```

Not a functional interface

```
Interface test{
 void display();
int add(int x, int y);
}
```

Object class

You have to get the benefit of Object methods only by overriding.

1. toString() → Print the status of the object
2. equals() → To compare two objects are equal
3. hashCode() → Will discuss in Lesson-11
4. clone() → Not covered

When you have an Inheritance for equals()

Case 1: You should have only one equals() from the parent, the comparison depends on parent attributes, you can use instanceof statategy. PersonwithJob is a Person.

Case 2: Same class Strategy, you should have two equals - one parent and the child. You can compare Person with Person type and PersonwithJob with PersonwithJob.

Drawback – Create asymmetric equality. (I can't compare Person and PersonwithJob) it breaks the rule of IS-A Rule. Personwith Job is a not Person.
In both cases you have to make the sub class as final, to avoid confusion in the multi level inheritance

Case 3: Composition (Has-A Relationship)

Cloning Types – Refer the Appexdix PPT for more information

1. Shallow clone
2. Deep clone.

Inside your class there is no composition – No worries you will get a copy.
When have a composition need to do deep copy.