EU8-JAVA-OOP Concepts (Part 1)

Encapsulation (Data Hiding)

▼ What is Encapsulation?

*Hiding the data by giving **private** access modifier*.

▼ How we perform encapsulation? (PIQ)

“private variables, public getter, public setter”

- Make all class fields ***private***
- Create ***public setter*** method to write data
- Create ***public getter*** method to read data

▼ Example

public class Car { // encapsulation

```
private String color, model, make;
private int year;

public String getColor() {
  return color;
}
public void setColor(String color) {
  this.color = color;
}
```

▼ if we generate the constructor, do we need the setter method? Answer: if you want to make your class immutable then you can just have getter.

Why we have both getter / setter?

▼ If we just want to get (read-only) or just want to change (write-only) data

- We can provide ***only getter*** in a class to make the class attribute ***(Read only)***

▼ Read only example

public class Car { // read-only example

```
private String color, model, make;
private int year;

public Car(String color, String model, String make, int year) {
  this.color = color;
  this.model = model;
  this.make = make;
  this.year = year;
}

public String getColor() {
  return color;
}

public String getModel() {
  return model;
}

public String getMake() {
  return make;
}

public int getYear() {
  return year;
}
```

- We can provide ***only setter*** in a class to make the class attribute ***write-only.***

Inheritance

▼ OOP inheritance builds ***Is A*** relationships between classes (super / sub relation)

- *Efficiency*
- *Less code, less duplication, less memory*
- *Easy to manage*

▼ How we perform inheritance? (PIQ)

- *One class inherits* ***the features (fields and methods)*** *of another class.*
- ***Parent (****also called* ***Super or Base) Class:*** The class whose features are inherited is known as super class (or a base class or a parent class)
- ***Child (****also called* ***Sub or Derived) Class:*** The class that inherits the other class is known as sub class (or derived class, or child class). *The subclass can add its own fields and methods in addition to the superclass fields and methods.*

The keyword used for inheritance is

extends

▼ What can be inherited?

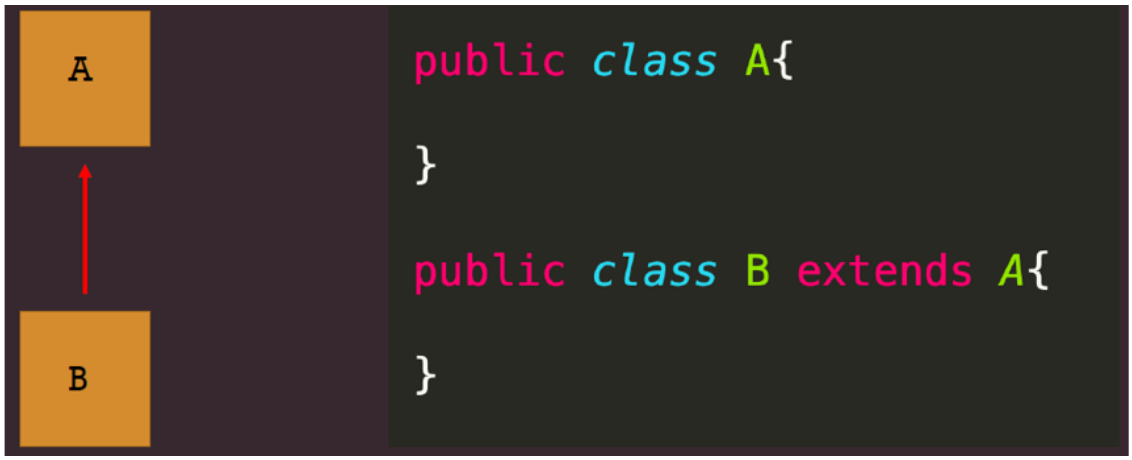
- All ***public variables and methods***.
- All ***protected*** variables and methods.
- All ***default variables and methods*** can be inherited only if super class and sub class are ***in the same package***.

▼ What cannot be inherited?

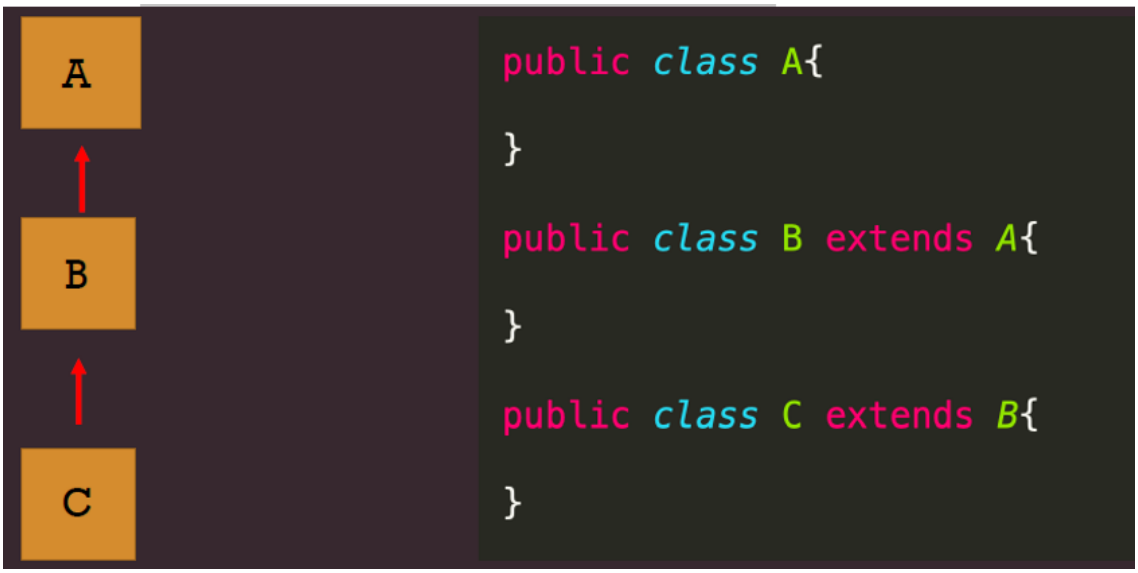
- ***Final*** class cannot be inherited, and it is immutable.
- ***Private variables and methods cannot be inherited***. But it is ***accessible using public getter/setters***.
- ***Constructors*** *cannot be inherited.*

▼ What are types of Inheritance?

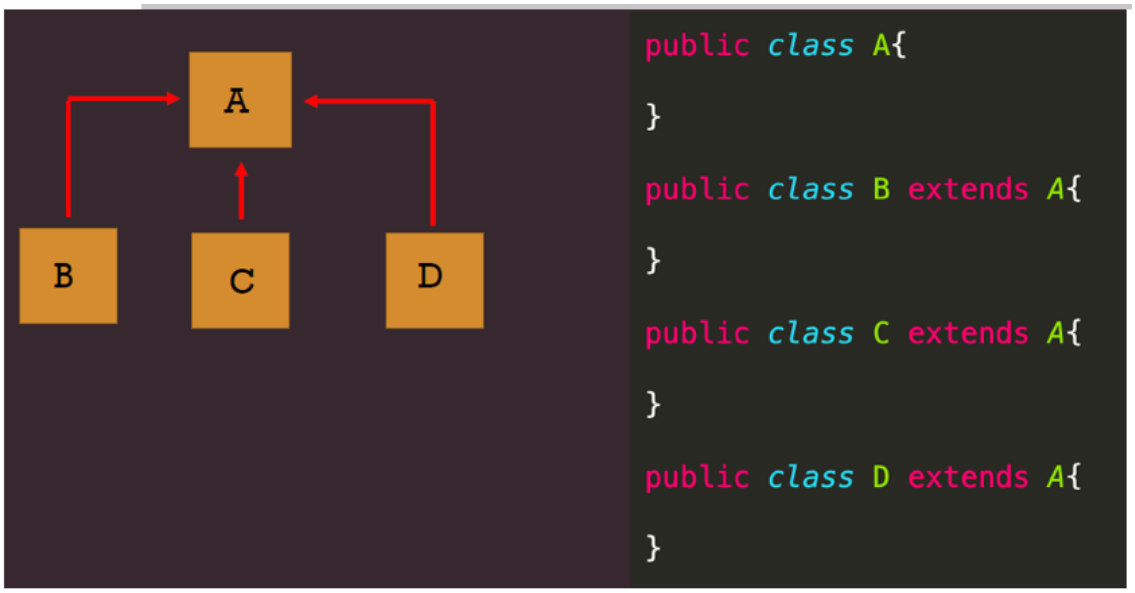
▼ ***Single Inheritance***: Subclasses inherit the features of one superclass.



▼ ***Multi-Level Inheritance:*** Subclass will be inheriting a SuperClass and as well as the subclass also act as the SuperClass to the other class.



▼ ***Hierarchical Inheritance:*** Once class serves as superclass for more than one sub class.



▼ ***Multiple Inheritance: Does JAVA support within classes? (PIQ)***

- Java DOES NOT support multiple inheritance ***with classes***. One class cannot have more than one superclass. But it can implement more than one interface.

java does not support multiple inheritance in case of class, but by using *interface* ***it can achieve multiple inheritance.***

▼ Superclass's Constructor

- the ***superclass constructor always executes before the subclass constructor***.
- The ***super()*** keyword refers to an object's superclass.
- If a subclass constructor does not explicitly call a superclass constructor, Java will automatically call the ***superclass's default constructor, or no-arg constructor***, just before the code in the subclass's constructor executes.
- It needs to be the ***first statement*** in the child class constructor
- ***this()*** also needs to be the first statement in the constructor, so ***super() and this() cannot be in the same constructor***
- If parent class only has constructor with parameters, then child constructor MUST make a matching super(params) call.

▼ static member’s inheritance

- Static members from a super class are inherited as long as ***access modifier allows***
- Static variables are shared class variables, it will have ***single central value*** for all objects and sub classes.
- Static methods, can be called either using ParentClass.methodName or SubClass.methodName
- Example

```
public static void main(String[] args) {
Car.getYear();
BMW.getYear();   }
```

▼ What is method overriding?

- Sometimes a subclass inherits a method from its superclass, but the method is ***inadequate*** for the subclass’s purpose.
- Because the *subclass is more specialized than the superclass*, it is sometimes necessary for the subclass to replace inadequate superclass methods with more suitable ones.

This is known ***overriding***.

▼ Method Overriding Rules / Overriding Superclass Methods

Overriding: *Declaring a method in subclass which is already present in the parent class.* Giving different implementations to the method.

1. There must be ***is-a relationship*** (inheritance)
2. The child method must have the ***same name*** as in the parent class
3. The child method must have the ***same parameter*** as in the parent class
4. ***Child Access modifier***: Needs to be ***same or more visible***
    a. public - > public
    b. protected - > protected, public
    c. default - > default, protected, public

5. ***Return type***:
    - (primitive) must be same or
    - (object) covariant type (same class type or sub class type)

▼ Only the instance method can be overridden

(Static method cannot be overridden but can be hidden)

- Private and final method cannot be overridden
- Protected method can be overridden

▼ Overloading vs. Overriding

Method Overloading	**Method Overriding**
Method overloading is performed ***within class***	Method overriding occurs ***in two classes*** that have ***IS-A relationship***
In case of method overloading, ***parameter must be different***	In case of method overriding, parameter ***must be same***
Access specifier can be changed	Access specifier ***must be same or more visible*** than original method
Return type of method ***does not matter*** in case of method overloading, it can be same or different	Return type ***must be same*** (for primitive and void) or covariant (for object) in method overriding
private and final methods can be overloaded	***private and final methods cannot be overridden***
A static method can be overloaded.	A static method ***cannot be overridden but can be hidden*** by defining a ***static*** method with same name and parameters in child class.

▼ super keyword in Java

The super keyword in java is a reference variable that is used to refer parent class objects. The keyword ***"super"*** is used with the concept of inheritance.

- Super with variables (super.variableName)
- Super with methods (super.methodName)

- Super with constructors (super())

▼ Inheriting Static Variables & Methods

We cannot override static methods or instance variables; we can hide them.

▼ Hiding Variables

- Variable hiding happens when we define a variable with the same name as a variable in a parent class.
- This creates 2 copies of the variable within an instance of the child class: one instance defined for the parent reference, and another defined for the child reference.
- If you are referencing the variable from within the parent class, the variable defined in the parent class is used.
- If you are referencing the variable from within the child class, the variable defined in the child class is used.

▼ Hiding Static Methods

- We cannot *override static methods; we can hide them*.
- A *hidden method occurs when a child class defines a static method with the same name and signature as a static method defined in a parent class*.
- ***Method hiding is similar but not exactly the same as method overriding.***
- The four rules for overriding a method must be followed when a method is hidden. In addition, a new rule is added for hiding a method, namely that the usage of the ***static keyword must be the same between parent and child classes*.**

Final Keyword

▼ What is “final” keyword?

“final” is a *non-access modifier* (specifier) which is applicable to a *variable, method, and class*.

Final variable : creates constant variable

Final method : prevent method overriding

Final class : prevent inheritance and make the class immutable

▼ final variables

Once it's initialized, its value is constant / unchangeable

▼ How to initialize ***'local final variables'***?

- if we don’t use it, initialization is not mandatory.
- If we use it, it can be initialized either on the same line or on a different line just before using it

▼ How to initialize ***'instance final variables'***?

- initialization is mandatory
- We can initialize instance final variables:
  - On the ***same line*** (same statement)
  - Inside a ***constructor***
  - Inside an ***instance (init) block*** *[init block : similar to static block. But init block only executes, when we create object and before constructor. If you don’t create an object, init block doesn’t run]*

▼ ***Can we initialize a final instance variable on another line?***

No, we cannot. It'll give a compile error.

▼ ***Can we initialize a final instance variable inside a static block?***

No, we cannot. Because static block cannot access to instance members

▼ ***How to initialize 'static final variables'?***

- initialization is mandatory
- We can initialize static final variables:
  - On the ***same line***
  - Inside the ***static block***

▼ ***Can we initialize a final static variable on another line?***

No, we cannot. It'll give a compile error.

▼ ***Can we initialize a final static variable inside a constructor or instance block?***

- No, we cannot. Because:
  - A static final variable must be initialized, it cannot be blank
  - But constructors and instance blocks are never called if we don't create an object.
  - That means, if we don't create an object, this static final variable may not be initialized.

▼ final arrays and final arrayList

- We ***can change the elements*** of a final array or final ArrayList without any problem
- Arrays are objects and object variables are always references in Java. When we declare an object variable as final, it means that the variable cannot be changed to refer to anything else. THIS VARIABLE CAN NOT ***POINT/RE-ASSIGN ANY OTHER OBJECT***.

▼ final methods

- When a method is declared with final keyword, it is called final method.
- A final method ***can be overloaded*** but ***cannot be overridden***.

▼ final class

- When a class is declared with final keyword, it is called a final class. A final class ***cannot be extended (inherited).***
- ***You cannot make a class immutable without making it final. (String class)***

Access Modifiers

▼ Summary Table for access modifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y