INHERITANCE POLYMORPHISM

CHAPTER 4

Objective

At the end of this presentation, you will be able to :

- Describe the relationship between classes in Object Oriented Pragramming (4.1.1)
- Create the multiplicity of relation in real-life situation in Class Diagram (4.1.2)

INTRODUCTION

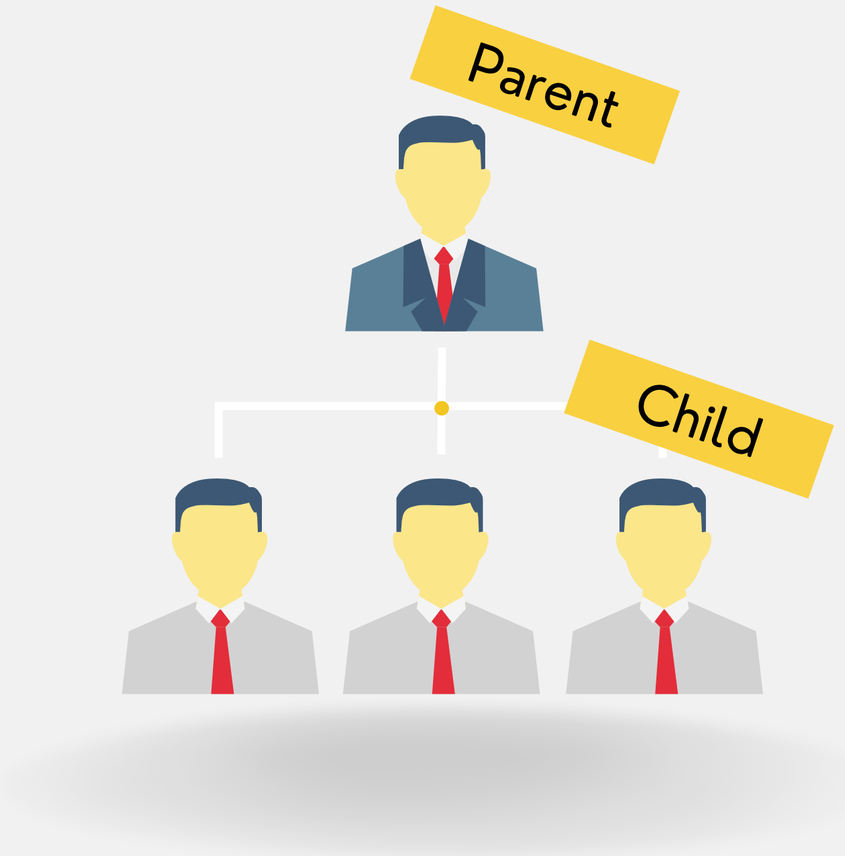
- Code reuse is one of advantage in OOP
- This reusability is possible due to the relationship between the classes
- There are 4 type of relationship:

  - Is-A
  - Part-Of
  - Has-A
  - Uses-A

All these relationship is based on inheritance, composition, association, aggregation and dependence

INHERITANCE

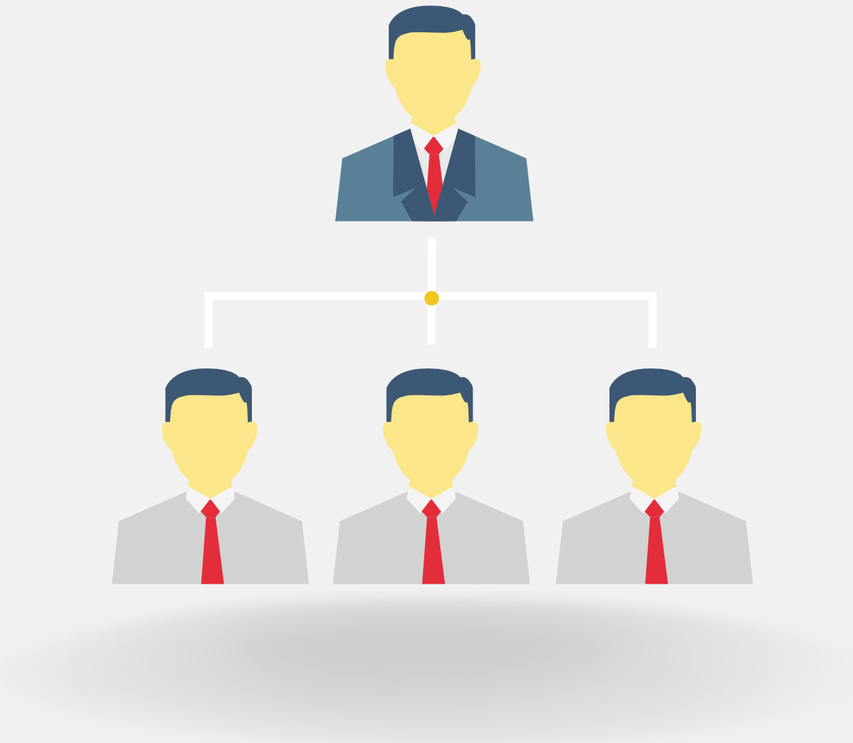
- Inheritance is a relationship that is "IS-A"
- The "IS-A" relationship is based entirely on inheritance, which can take two forms:
  - Class Inheritance and Interface Inheritance.
  - Inheritance is a parent-child relationship in which we use existing class code to construct a new class.
  - For example:
    - A is a sort of B



INHERITANCE

Take a look at a real-life scenario for a better understanding:

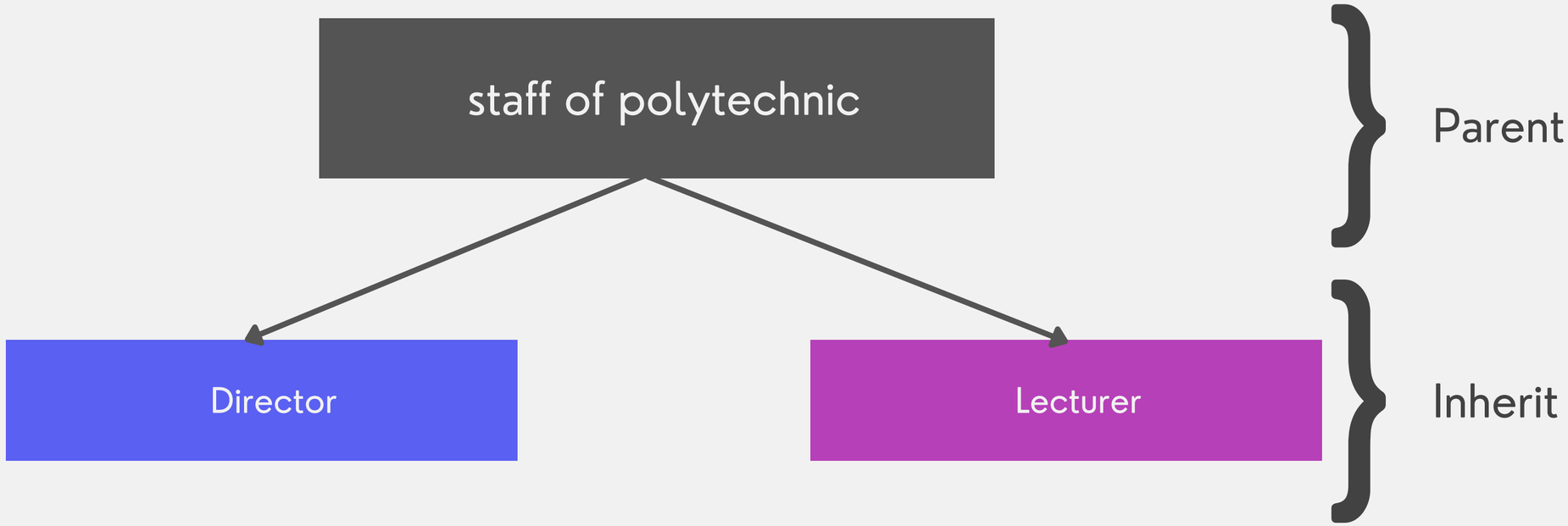
- Director is staff of polytechnic
- All lecturers are staff member of polytechnic
- Director and lecturers has thumbprint id for proof attendance into polytechnic
- The Director is responsible for overseeing the lecturer's work in order to complete the course in the time allocated.



INHERITANCE

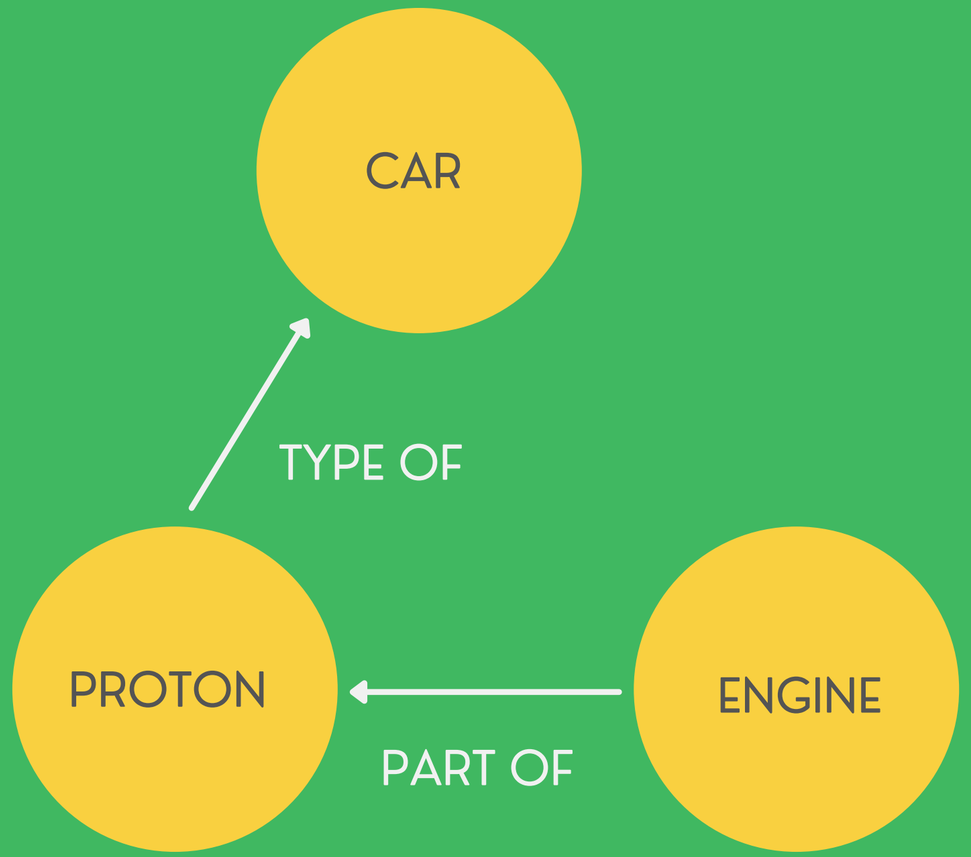
Let's start from the first two assumptions:

1. Director is staff of polytechnic
2. All lecturers are staff of polytechnic



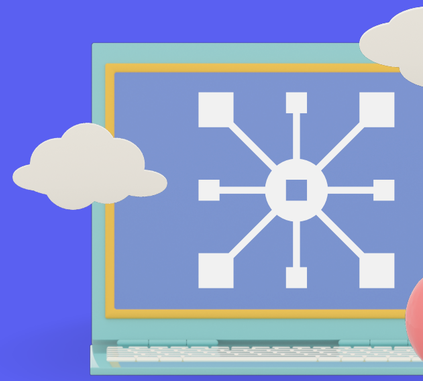
COMPOSITON

- A "part-of" connection is composition.
- It refers to the use of instance variables that point to other objects.
- Both entities are interdependent in a composition connection,
  - For example:
    - the engine is part of the car,
    - the heart is part of the body.

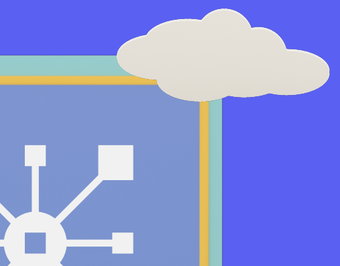


Engine is a part of each car and both are dependent on each other.

ASSOCIATION

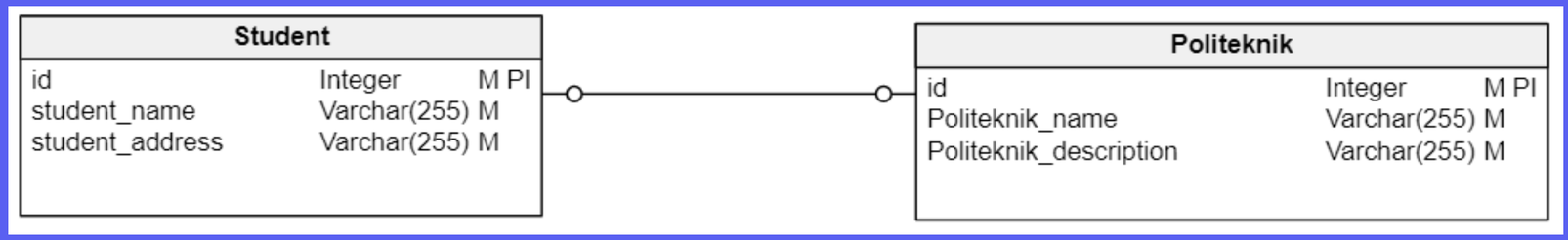


- Refers to a "has-a" relationship.
- Through their objects, associations establish the relationship between two classes.
  - One-to-one [1:1]
  - one-to-many [1:N]
  - many-to-one [N:1]
  - many-to-many [N:N]
- If we have two classes, for example:
  - Both of these entities are said to have a "has-a" relationship, if they share each other's object for some task and at the same time they can exists without each others dependency or both have their own life time.

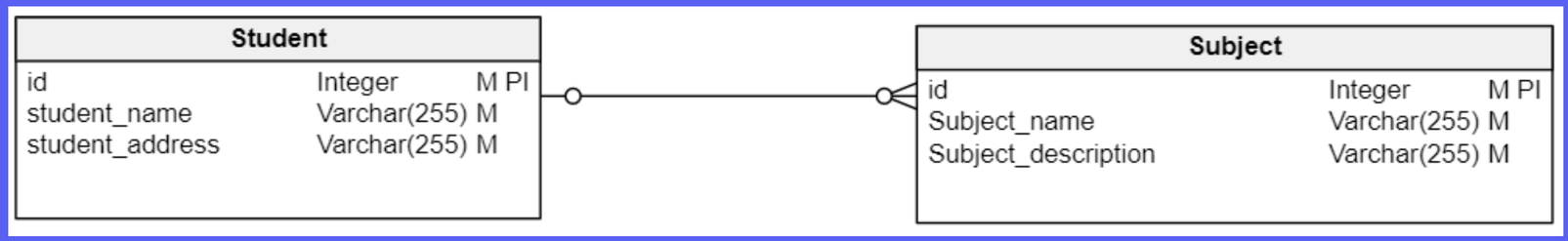


ASSOCIATION

- ONE-TO-ONE

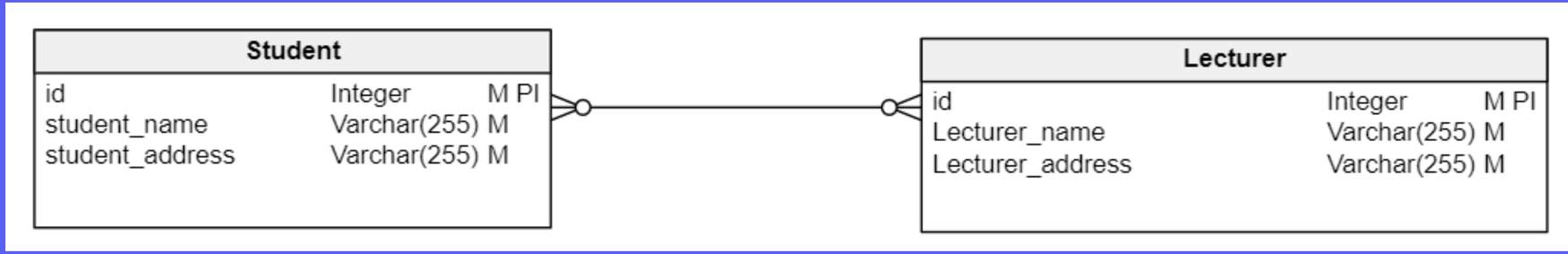


- ONE-TO-MANY



ASSOCIATION

- MANY-TO-MANY

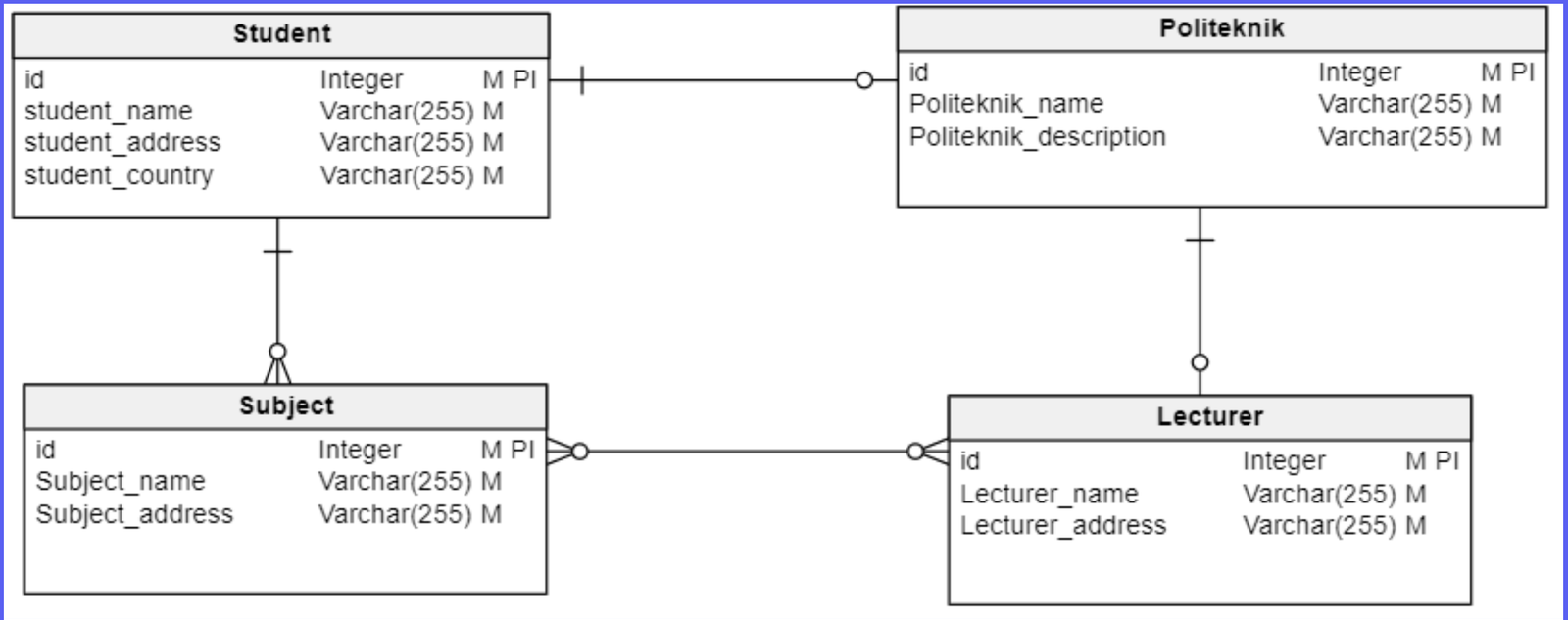


RECAP: Try to implement one-to-one, one-to-many and many-to-many relationship into your real life. (Family?, Work Place?, Order Food?, Shopping?, Bank?)

10Mins

ASSOCIATION

- MULTIPLICITY OF RELATIONSHIP IN CLASS DIAGRAM



- One politeknik for each of student at the same time to study
- Each lecturer must lecture at one politeknik
- One student has many subject at one semester
- Many subjects for many lecturers to conduct for semester

ASSOCIATION

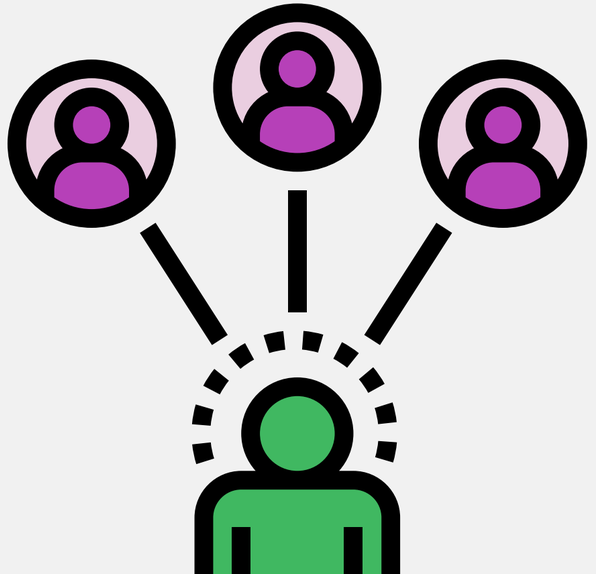
- MULTIPLICITY OF RELATIONSHIP IN CLASS DIAGRAM

RECAP: Try to implement mutiplicity of relationship based your situation (Family?, Work Place?, Order Food?, Shopping?, Bank?)

10Mins

AGGREGATION

- Aggregation is the "has-a" relationship.
- It is a type of relationship that is unique.
- There are no classes (entities) that work as owners in association, but one entity works as owner in aggregation.
- In aggregation, two entities meet for a short period of time to do some work before being separated.
- Aggregation is a one-sided relationship.



DEPENDENCE

- This relationship refers to a "Uses-A" relationship.
- It create an object of one class inside a method of another class.
- In other words, dependency in Java refers to when a class's method uses an object from another class.
    - In Java, it is the most evident and general relationship.

DEPENDENCE

This figure:
A method display() of class B uses an object of class A. So, class A depends on another class B if it uses an object of class A.

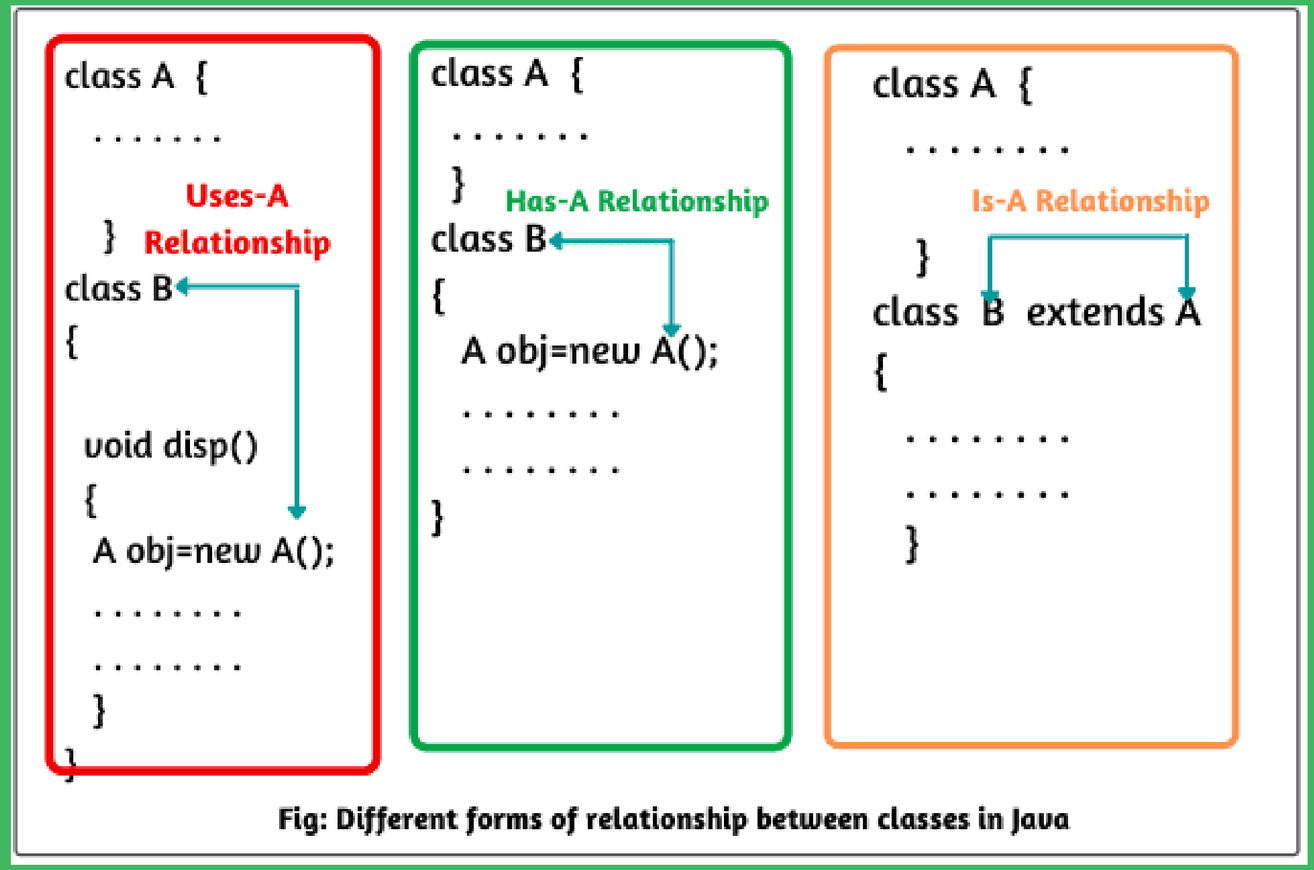


Fig: Different forms of relationship between classes in Java

Summary

- IS-A relationship based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance.
- PART-A relationship is manage to use one class as references instead to refer all classes.
- HAS-a relationship is composition relationship which is a productive way of code reuse.
- USE-A relationship is make less dependence which is a good programming practice because too much dependencies make system diffucult to manage.

Programming Language (JAVA)

Chapter 3 : Inheritance

Presentation 1

Revision

1. Define Function.
2. Write the syntax and example for defining a function.
3. Write the syntax for calling a Function.
4. Define Polymorphism.
5. Name the 2 ways of implementing Polymorphism in Java.

Objectives

At the end of this presentation, you will be able to :

- Describe Inheritance in Java program.
- Apply keyword super in Java program.

Need for Inheritance

```
class Student
{
  String Name;
  int Age;
  char Gender;
  double mark;
```

```
void getdetails( )
  {
    -----
    -----
  }
}
```

Need for Inheritance (Contd..)

```
class Teacher
{
  String Name, subject;
  int Age;
  char Gender;
  double Basic;
```

```
  void getdetails( )
  {
    -----
    -----
  }
}
```

Advantages of Inheritance

The advantages of Inheritance are:

1. The variables and methods that are common to two or more classes can be defined in one class and used in other classes. This is called as code reusability.
2. When there is any modification in the common information, it will apply to all the inherited classes.
3. Inheritance avoids code redundancy.

Types of Inheritance

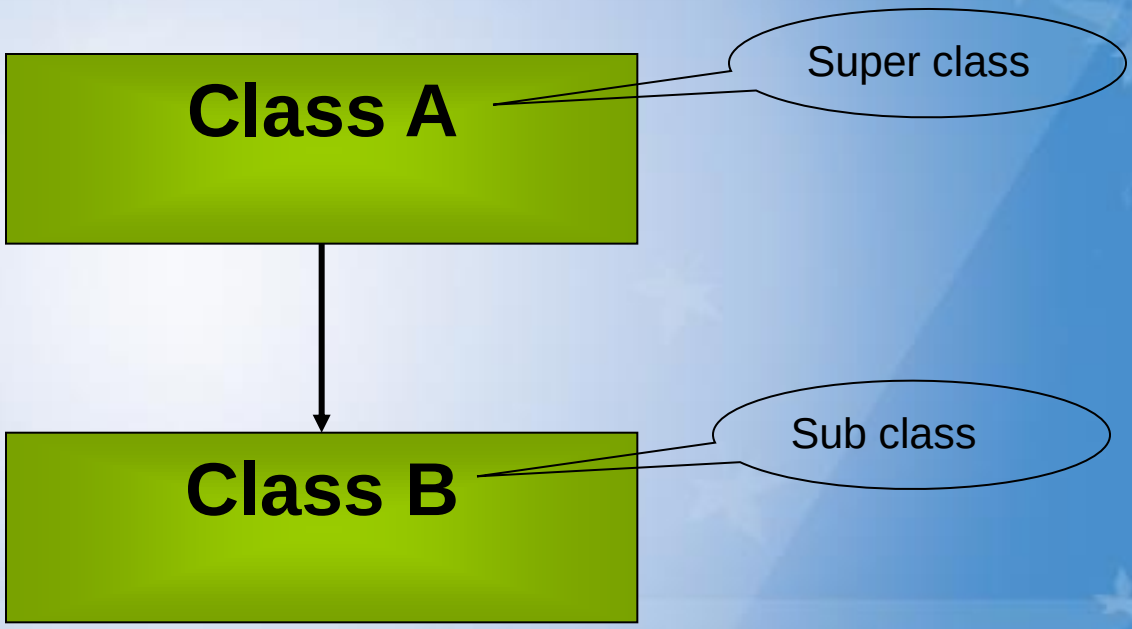
There are 3 types of Inheritance. They are:

1. Single Inheritance
2. Multiple Inheritance and
3. Multi-Level Inheritance

Single Inheritance

- When a class is inherited from only one existing class, then it is *Single inheritance*.
- The inherited class is a *sub class* and the existing class from which a sub class is created is the *super class*.
- In single inheritance a super class can have any number of sub classes but a sub class can have only one super class.

Single Inheritance

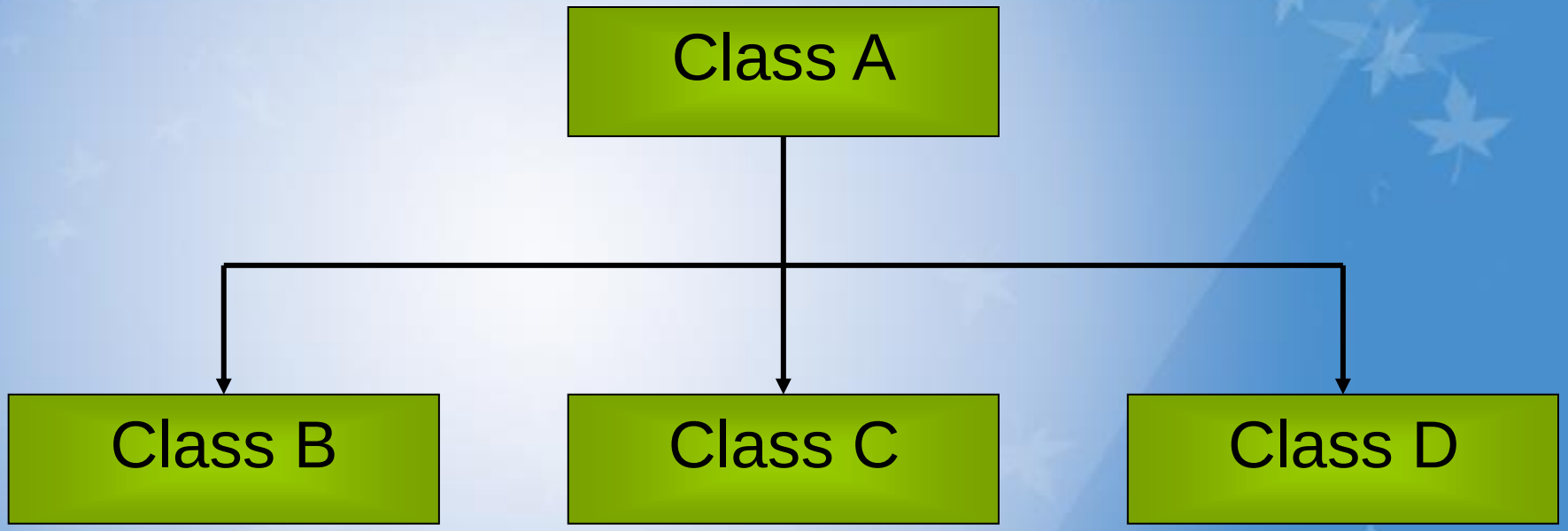


Single Inheritance - Code

```
class A
{
-------
-------
}
```

```
class B extends A
{
-------
-------
}
```

Single Inheritance



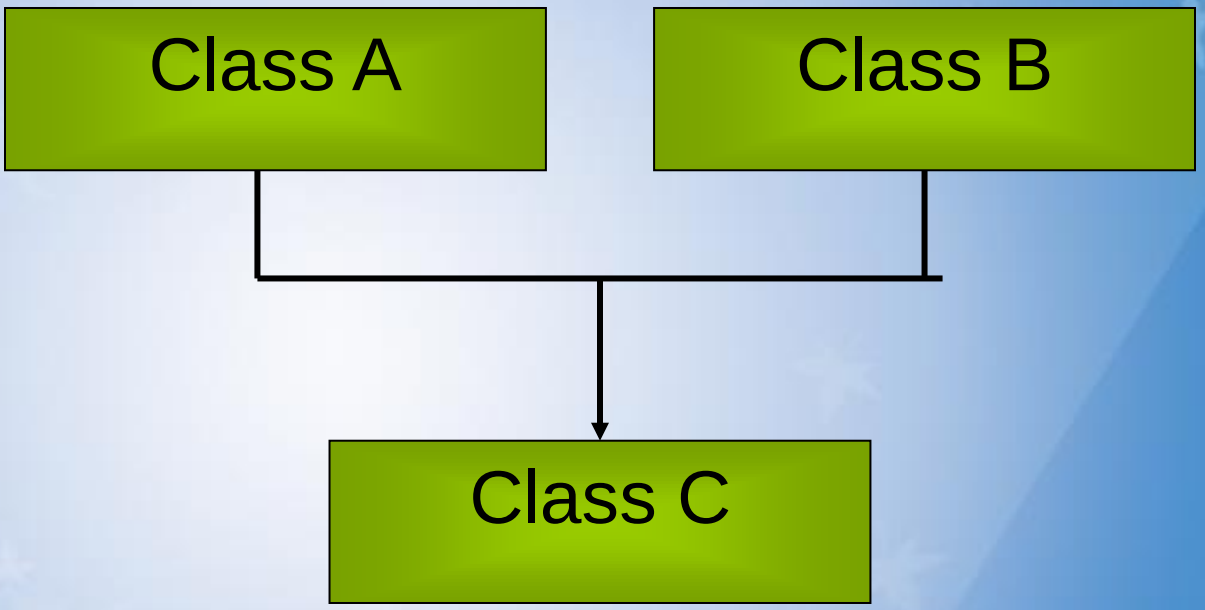
Single Inheritance - Code

```
class A
{
-------
-------
}

class B extends A
{
-------
-------
}

class C extends A
{
-------
-------
}

class D extends A
{
-------
-------
}
```

Multiple Inheritance

- When a class is inherited from more than one existing class, then it is *Multiple inheritance*.
- The inherited class is a sub class and all the existing classes from which the sub class is created are super classes.
- Java supports Multiple Inheritance through a concept called *Interface*.

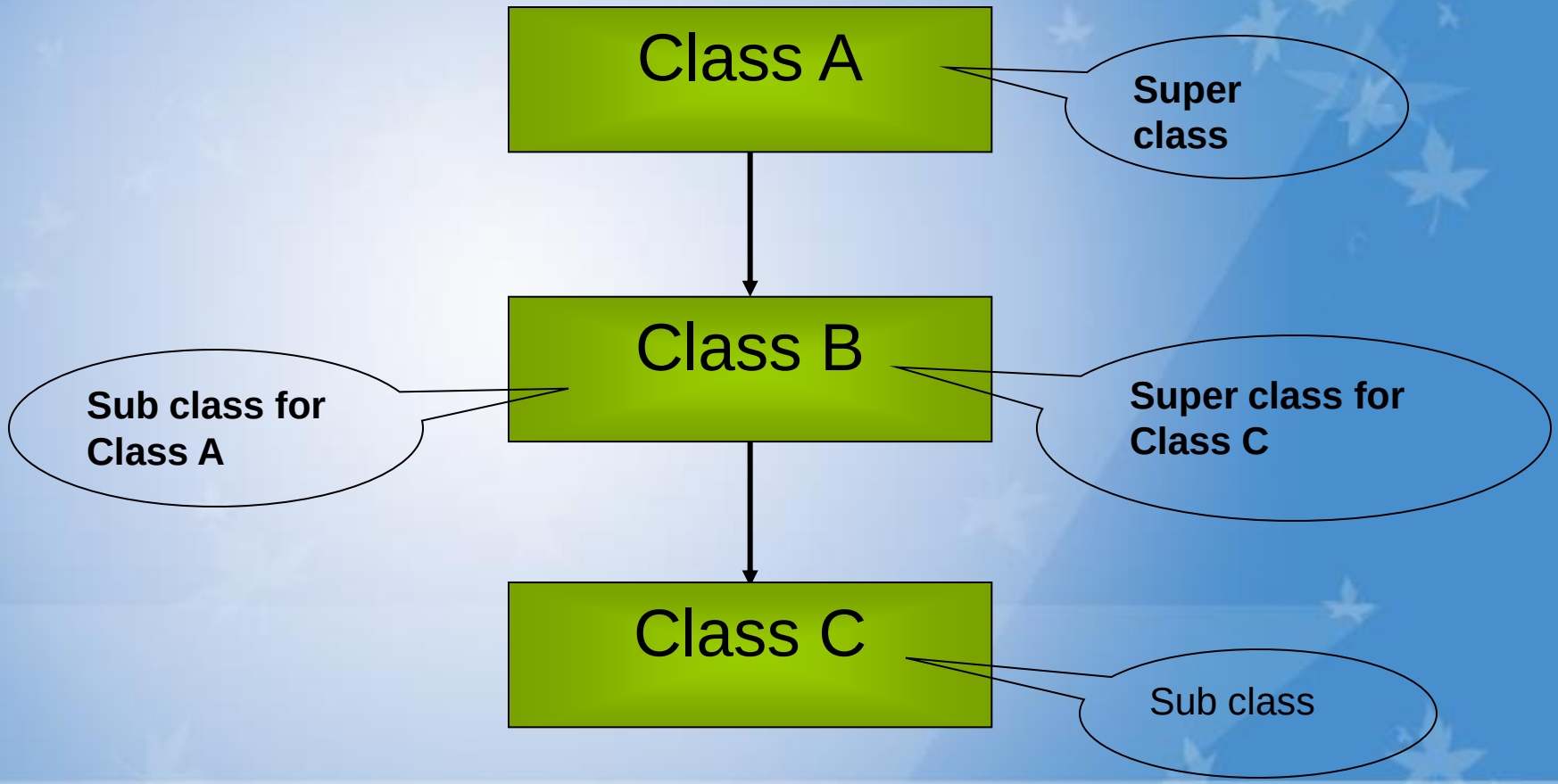
Multiple Inheritance



Multi-Level Inheritance

- Multi-Level Inheritance is the extension of Single Inheritance.
- When a class is inherited from another sub class, then it is *Multi-Level Inheritance*.
- The sub class at the lowest level can access the member variables and methods of all the super classes at the higher level.

Multi-Level Inheritance



Multi-Level Inheritance - Code

```
class A
{
-------
-------
}
```

```
class B extends A
{
-------
-------
}
```

```
class C extends B
{
-------
-------
}
```

Super Class

- A new class can be inherited from an existing class.
- The existing class from which a new class is created is known as the *Super class*.
- It is also called as *Base class*.

SubClass

- The new class that is inherited from an existing class is known as *Subclass*.
- This class is also called as *Derived class*.
- The subclass inherits all the accessible methods and member variables of the super class.
- A subclass is created using the *extends* keyword.

Creating SubClass

Syntax

class <class_name> extends
<existing_class_name>

{

}

Example

class Cylinder extends Sphere

{

}

- An object of the sub class can access the variables and methods of the super class.

Accessing objects of super class

- An object of the subclass can access the variables and methods of the super class using the *dot* operator.
- **Syntax** for accessing a super class variable and method:

Object.variable

Object.method()

Hands-On!

- Program *Shape.java* illustrates how an object of a subclass can access the variables and methods of a super class.

Super Class and SubClass

- When an object of a super class is created, it has only the variables declared in its own class.
- When an object of a subclass is created, the subclass object has a copy of the variables defined in its super class along with the variables of its own class.

The super keyword

- The super class variable, method and constructor can be accessed from a subclass using super keyword.
- The super class method and variable can be accessed using the dot operator as given:

super.variable_name

super.method_name

Hands-On!

Program *Sup_vol.java* illustrates how a subclass can access a super class variables and methods.

Super Class and SubClass

- The super class constructor can be invoked from a subclass using the *super* keyword.
- The super class variable and method can be accessed from a subclass using the *super* keyword followed by *dot* operator and the variable name or method name.

Activity 6.8.1

Detect the errors in the following program:

```
// Program to display student details using
      inheritance

class Person

{

  String name;

  int age;

}
```

Activity 6.8.1 (Contd..)

```
class Student inherit Person
{
  int mark1,mark2,mark3;
  void putdata()
  {
    System.out.println("Name = " + name);
    System.out.println("Age  = " + age);
   System.out.println("Mark1= " + mark1);
```

Activity 6.8.1 (Contd..)

```
    System.out.println("Mark2= " + mark2);
    System.out.println("Mark3= " + mark3);
  }
}
class Marks
{
public static void main(string args[])
  {
```

Activity 6.8.1 (Contd..)

```
obj1 Student=new Student();
obj1.name=Sultana;
obj1.age=17;
obj1.mark1=67;
obj1.mark2=87;
obj1.mark3=97;
putdata();
}
}
```

Lab Exercise

1. Write a program to define a class *Marks* with data members *Mark1, Mark2 and Mark3*. Create a sub class *Total* of *Marks* class that has a method to find the total of the 3 marks

2. Write a program to define a class *Box* with data members *width, depth and height*. Create a sub class *Volume* of *Box* class that has a method to find the volume of the box.

Lab Exercise (Contd..)

3. Write a program to define a class *Car* with data members *Car_No, Avg_Speed* and *Rank*. Create a sub class of Car class that has a method to display the values of the member variables.

Summary

In this presentation, you learnt the following

- *Inheritance* avoids code redundancy because the extended classes need not have the code that is shared with other classes.
- There are three types of Inheritance are *Single Inheritance, Multiple Inheritance* and *Multi-Level Inheritance.*
- When a class is inherited from only one existing class, then it is *single inheritance*.

Summary

In this presentation, you learnt the following

- When a class is inherited from more than one existing class, then it is *Multiple inheritance*.
- When a class is inherited from another sub class, then it is *Multi-Level Inheritance*.
- Whe existing class from which a new class is created is known as the *Super class*.

Summary

In this presentation, you learnt the following

- The new class that is inherited from an existing class is known as *Sub class*.
- A sub class is created using the *extends* keyword.
- An object of the sub class can access the variables and methods of the super class using the *dot* operator.

Assignment

1. Define *Inheritance*.
2. List the advantages of *Inheritance.*
3. Write the syntax to create a *subclass*.
4. Write the syntax for an object of a *subclass* to access the variables, methods of a *super class.*

Programming Language (JAVA)

Polymorphism

LESSON LEARNING OUTCOMES :

At the end of this presentation, you will be able to:

- Implement the concept of polymorphism
- Overload methods in programs
- Overload constructors in programs
- List the advantages of overloading methods and constructors

POLYMORPHISM

- Polymorphism is the existence of an entity in many forms. In object oriented programming language it refers to the ability of the objects to behave differently based on the input given.
- Polymorphism is implemented in Java using techniques:
  - **Method Overloading (Static-compile time)**
  - **Method Overriding(Dynamic-run time)**

    Example:

    A reference variable of the super class can refer to a sub class object

    Doctor obj = new Surgeon();

POLYMORPHISM EXAMPLE

```
class Shape
{
    public void draw() {
    }
}
class Square extends Shape {
   public void draw() {      // <-- overridden method
   }
   .   // other methods or variables declaration
 }
class Circle extends Shape {
   public void draw() {      // <-- overridden method
   }
    .    // other methods or variables declaration
}
class Shapes {
   public static void main(String[] args) {
      Shape a = new Square();       // <-- upcasting Square to Shape
      Shape b = new Circle();       // <-- upcasting Circle to Shape
      a.draw();          // draw a square
      b.draw();          // draw a circle
   }
}
```

METHOD OVERLOADING

- Two or more methods having the same name but different signature.
- Methods which have same name but different parameters are called as overloaded methods.
- The parameters in overloaded methods should differ in at least one of the following:
  - Number of parameters
  - Data type of the parameters

HANDS-ON!

- Program m_o_load.java will illustrates method Overloading.

ACTIVITY (A)

```
class Over
{
	void display()
	{
		System.out.println("Melaka ");
	}

	void display(int i)
	 {
		 int j=0;
		while(j<i)
		 {
				System.out.println("Johor ");
				 j++;
		}
	}
	 void display(String str,int i)
	 {
		 for (int j=1;j<=i;j++)
			System.out.println(str);
	}
}
```

```
class Poly {
	public static void main(String args[])
	{
	  Over obj=new Over();
	  obj.display();
	  obj.display(2);
	  obj.display("Kuala lumpur ",3);
	}
}
```

ACTIVITY (B)

1. Fill in the blanks in the following program that uses overloaded methods:

```
// Program to multiply numbers using overloaded methods

class Product
{
   public ______ void main(String args[])
    {
            int result1;
            double result2;
            prod obj=new _____();
            result1=obj.multiply(12,13);
            result2=obj.multiply(2.5,3.2);
            _______.out.println(result1);
            System.out.println(result2);
    }
}
```

ACTIVITY (B) (CONTD…)

```
class prod
{
    ______ multiply(int No1, int No2)
     {
              return( No1 * No2 );
    }

     double ________(double No1, ________ No2)
    {
        return( No1 * No2 );
    }
}
```

CONSTRUCTOR OVERLOADING

- The concept of having two or more constructors with different signature in the same class is called *Constructor Overloading*.
- Two or more constructors in a class with same name and different parameters are called as overloaded constructors.

HANDS-ON!

- Program c_o_load.java illustrates constructor overloading.

HANDS-ON!

- Perform adv_mtdovrl.java to explain the advantage of method overloading.

HANDS-ON!

- Perform adv_consovrl.java to explain the advantage of constructor overloading.

ACTIVITY (A)

1. The following program displays the addition of 2 integers and 2 real numbers. Insert overloaded constructors in the blanks given in the following program:

ACTIVITY (A) (CONTD…)

```
class Numbers
{
    public static void main(String args[])
    {
        Sum obj1=new Sum(0,0);
        Sum obj2=new Sum(0.0,0.0);
        obj1.add(5,7);
        obj2.add(5.0,7.0);
     }
}
```

ACTIVITY (A) (CONTD…)

```
class Sum
{
    _____(int a,int b)
    {
        ________
        ________
    }

    ___(double c,double d)
    {
        _________
        _________
    }
```

```
    void add(int a,int b)
    {
        this.a=a;
        this.b=b;
        System.out.println
            ("The sum is " + (this.a + this.b));
    }

    void add(double c,double d)
    {
        this.c=c;
        this.d=d;
        System.out.println
            ("The sum is " + (this.c + this.d));
    }
}
```

SUMMARY

In this presentation, you learnt the following

- Define Polymorphism.
- Name the 2 ways of implementing Polymorphism in Java.
- Define Constructor Overloading.

LAB EXERCISE

6. Write a program using overloaded methods to print a multiplication table for an integer and a real number.
7. Write a program to print the absolute value of an integer and a real number using overloaded methods.

ASSIGNMENT

1. Define Constructor Overloading.
2. Write a program to find the Simple Interest. Use constructors to initialize the values of Principal, Number of years and rate of interest to 0.0.

Programming Language (JAVA)

Chapter 3 : Inheritance

Presentation 2

Objectives

At the end of this presentation, you will be able to :

- Create Abstract classes.
- Override methods.
- Secure the data using Access specifiers.
- Modify the behavior of classes, methods, variables and constructors using Modifiers.

Abstract Class

- A class that cannot be instantiated and contains one or more abstract methods is called as *Abstract class*.
- A class can be instantiated, which means an object can be created for a class. Such classes are called as *Concrete classes*.
- The classes that cannot be instantiated are called as *Abstract classes*.

Abstract Class (Contd..)

- The abstract class gives a general structure for a class.
- This structure can be implemented by their sub classes in different ways.

Abstract Class (Contd..)

- Consider a situation, where all the sub classes of a class must implement a method of its super class. When you forget to implement the method in a sub class.
- In such cases, you can declare the super class and the method in the super class as abstract.
- Now, a compiler error occurs if you forget to implement all the abstract methods of the super class in your sub class.

Hands-On!

- Program *Abst.java* illustrates the use of Abstract classes in java.

Activity 2

1. Fill in the blanks in the following program that uses abstract class:

```
/* Program to find the volume of Sphere,
Cone and Cylinder using abstract class */

class Volume

{

double r,h;
```

Activity 2 (Contd..)

```
abstract void vol()
 {
 }
}
class Sphere extends Volume
{
```

Activity 2 (Contd..)

```
void ()
 {
  System.out.println("The volume of Sphere is
      " + (4.0/3.0*3.14*r*r*r));
 }
}
```

Activity 2 (Contd..)

```
class Cylinder extends _______
{
 void vol()
 {
  System.out.println("The volume of Cylinder
      is " + (3.14*r*r*h));
 }
}
```

Activity 6.8.2 (Contd..)

```
class Cone extends Volume
{
 void vol()
 {
  System.out.println("The volume of Cone is "
      + (1.0/3.0*3.14*r*r*h));
 }
}
```

Activity 2 (Contd..)

```
class Abst
{
 public static void main(String args[])
 {
  Sphere obj1=new Sphere();
  obj1.r=10;
  obj1.vol();
  Cylinder obj2=new Cylinder();
  obj2.r=10;
```

Activity 2 (Contd..)

```
obj2.h=6;
  _____.vol();
  Cone obj3=____ Cone();
  obj3.r=10;
  obj3.h=6;
  obj3.vol();
 }
}
```

Lab Exercise

4. Write a program to define an abstract class *Shape* which has a method area() in it. Create two more classes *Rectangle* and *Square* that extends the *Shape* class. The class *Rectangle* should implement the method *area()* to find the area of rectangle. The class *Square* should implement the method *area()* to find the area of square.

Method Overriding

- *Method overriding* refers to the concept of the methods in the sub class and the super class using the same method name with identical signature.

Method Overriding (Contd..)

- The method in the sub class hides or overrides the method definition in the super class. This is known as *Method Overriding.*
- Method overriding can be used in situations, where you want a method in the super class to be implemented differently in the sub class but with same name and parameters.

Hands-On!

- Program *MtdOvrd.java* illustrates Method Overriding in java

Activity 3

1. Open the data file MtdOveride.java and write the output for the following program

```
/* To convert Fahrenheit into Centigrade and vice
   versa using overridden methods */

class Centigrade

{

 double f,c;
```

Activity 3 (Contd..)

```
void convert(double num)
 {
 this.f=num;
  c=(f-32.0)*5.0/9.0;
  System.out.println("The temperature in
      centigrade is "  + c);
 }
}
```

Activity 3 (Contd..)

```
class Fahrenhiet extends Centigrade
{
 void convert(double num)
 {
  this.c=num;
  f=9.0/5.0*c+32.0;
  System.out.println("The temperature in
      fahrenhiet is " + f);
 }
}
```

Activity 3 (Contd..)

```
class MtdOveride
{
 public static void main(String args[])
 {
  Centigrade obj1=new Centigrade();
  Fahrenhiet obj2=new Fahrenhiet();
  obj1.convert(98);
  obj2.convert(36.66);
 }
}
```

Lab Exercise

5. Write a program to perform addition, subtraction, multiplication and division using overridden methods.

Access Specifiers and Modifiers

- In some situations, the member variables or methods of one class may or may not be allowed to access other class members.
- The access of the members of a class can be controlled by *Access Specifiers*.
- The 3 access specifiers in Java are
  - *public*
  - *private*
  - *protected*.

Access Specifiers and Modifiers

- Public – When a class member is declared as *public*, it can be accessed by the code in any class.
- Private - When it is declared as *private* it can be accessed only by the code in the same class where it is declared.
- Protected- When it is declared as *protected*, it can be accessed by the code in same class and inherited classes.

Hands-On!

Program *AccessTest.java* explains the different access specifiers.

Class Modifiers

Modifier	Effect of the Modifier
abstract	The class declared as abstract cannot be instantiated.
final	The class declared as final cannot be extended.

Variable Modifiers

Modifier	Effect of the Modifier
static	Specifies that the variable is common to all the objects of that class.
final	Specifies that the value of the identifier cannot be changed.

Hands-On!

- Program *Modifier.java* illustrates the use of Modifiers in java.

Method Modifiers

Modifier	Function
abstract	Specifies that the method cannot be implemented in this class and it should be implemented in its sub classes.
final	Specifies that the method cannot be overridden.
static	Specifies that the method is common to all the objects. These methods can be accessed by using the class name.

Summary

In this presentation, you learnt the following

- The classes that cannot be instantiated are called as *Abstract classes.*
- The method in the sub class overrides the method in the super class. This is known as *Method Overriding.*
- The three access specifiers in Java are *public, private* and *protected*.

Assignment

5. Define *Abstract class* and state the advantage of using it.
6. Define *Method Overriding*.
7. List the method *modifiers* and their functions.


SLOW PROGRESS
IS BETTER
GYMQUOTES.CO
THAN NO PROGRESS.
STAY POSITIVE
AND DON'T GIVE UP.

Chapter 4 : INHERITANCE AND POLYMORPHISM

Apply the concept of an Interface

- Explain interface.
- Classify the built-in interface class in Java library.
- Create Java program using the following classes:
  - extending interface
  - implementing interface

Java Interfaces

- Interfaces **declare *features(i.e. methods)*** but provide ***NO* implementation**
- **classes** that implement an interface ***MUST* provide an implementation** for each feature
- Interfaces can inherit from another interface
  - at most one superinterface
- A Java class **can implement more than one interface**
  - Java inherits(extends) from at most one type, but can implement more than one interface.

Java Interfaces (cont)

- A special type of class - a "pure" abstract class:
- An interface declaration begins with the keyword '**interface**' and contains only **constants** and **abstract methods** (does not provide the implementation for the prototypes, only the definition (signature))

Java Interfaces (cont)

- Printable defines 3 methods
- Displayable inherits from Printable and adds some more method signatures
- A class that implements Displayable will have to provide an implementation for all 6 methods.

```
interface Printable
{
   public String printInstVar();
   public String showInfo();
   public String printWithSpaces();
}

interface Displayable extends Printable
{
   public String display();
   public String refresh();
   public void displayColor(Color c);

}
```

Java Interfaces (cont)

- keyword **implements** followed by a list of one or more comma separated interface names

- Method signatures **MUST** match
  - same modifiers
  - same method names
  - same number of arguments
  - corresponding argument types.

```
public class Circle implements Displayable
{
  private int radius;
  public String printInstVar()
  {  System.out.println("Radius "+radius);
  }
  public String showInfo()
  {  System.out.println("I am a Triangle
                    with radius"+radius);
  }
  public String printWithSpaces(){
  ...
  }
  public String display(){
  ...
  }
  public String refresh(){
  ...
  }
  public void displayColor(Color c){
  ...
  }
}
```

Types Revisited

- In Java, each interface defines a type. Interface extension and implementation as *subtype* relationships
- A subtype relation in Java is:
  - *if class C1 extends class C2 then C1 is a subtype of C2*
  - *if interface D_1 extends D2 then I is a subtype of I*
  - *if class C implements interface I then C is a subtype of I*
  - *for every interface I, I is a subtype of Object*
  - *for every type T , T[ ] is a subtype of Object*
  - *if T_1 is a subtype of T_2 then T_1[ ] is a subtype of T_2[ ]*

Types of Circle

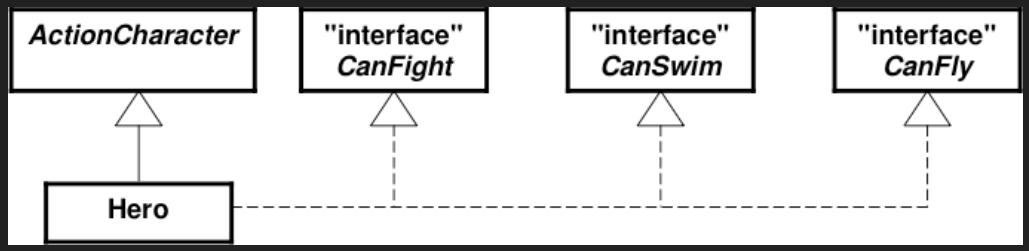
Circle is a subtype of

- Object
- Displayable
- Printable

```
public class Circle implements
    Displayable {
 private int radius;
 public String printInstVar(){
  System.out.println("Radius "
                          +radius);
 }
 public String showInfo(){
  System.out.println(" I am a
   Triangle with radius "+ radius);
 }
 public String printWithSpaces(){
 ...
 }
 public String display(){
 ...
 }
 public String refresh(){
 ...
 }
 public void displayColor(Color c){
 ...
 }

}
```

Implementing interfaces

- Implementations provide complete methods:




```
interface CanFight {
    void fight();
}
interface CanSwim {
    void swim();
}
interface CanFly {
    void fly();
}
class ActionCharacter {
    public void fight() {...}
}
```

```
class Hero extends ActionCharacter
implements CanFight, CanSwim, CanFly {
    public void swim() {...}
    public void fly() {...}
}
```

Inheritance and its forms

Combination

- child class inherits features from more than one parent
  - Java does not *directly* support this last form, although we can simulate it (more on this next time)

Using interfaces a class can inherit features from more that one parent.

- parents do not have to be in an direct inheritance relationship

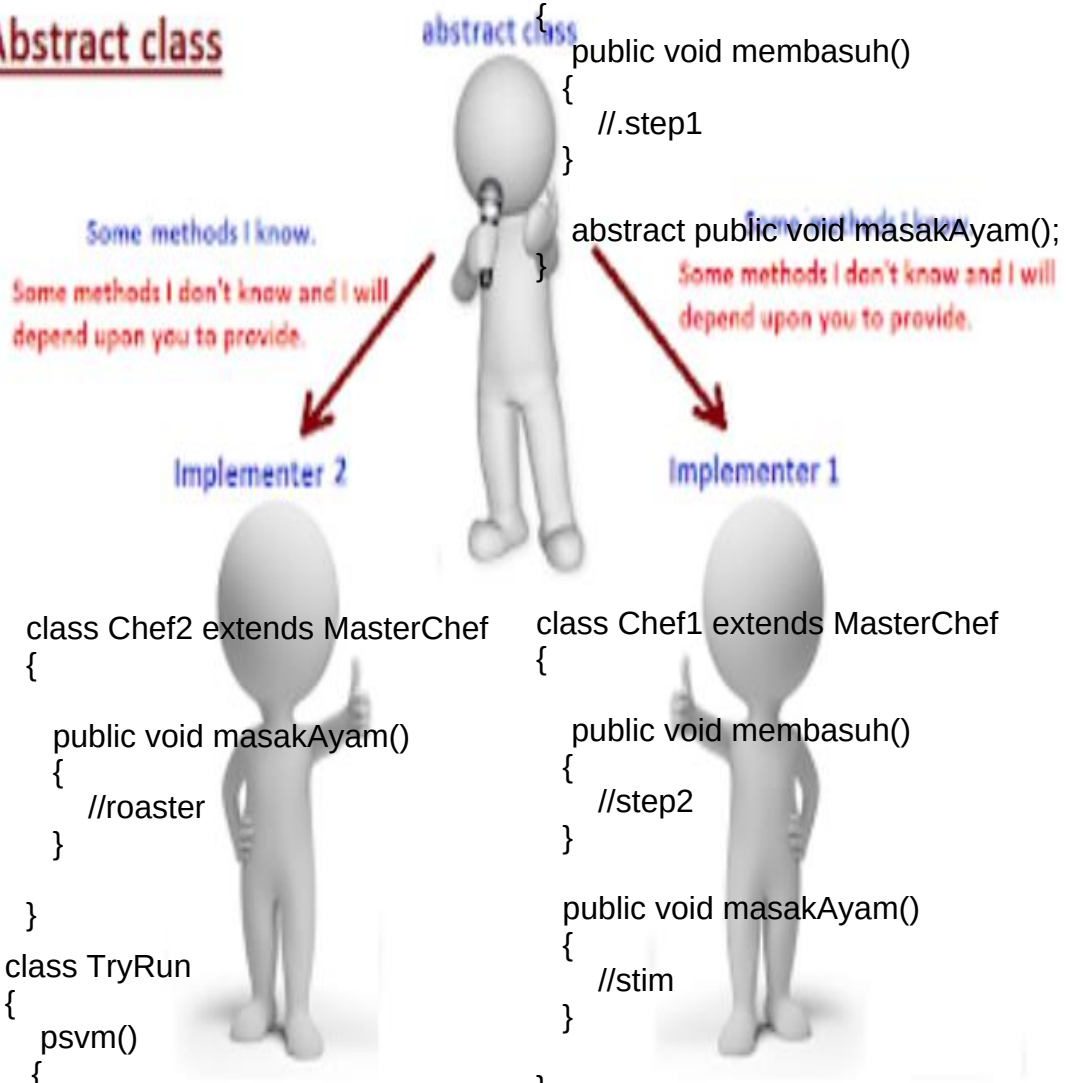
Interface vs Abstract class

- specify the form of a concept: not implementing it
- cannot have data members, only constants
- lightweight to implement
- multiple-implementations
- based on “has-a” (composition)

- incomplete class (may have partial implementations) which needs a specialization (derivation)
- can have data members
- base class: used to initialize a hierarchy of classes
- single-inheritance
- based on “is-a” (inheritance)

Interface

Abstract class




```
abstract class MasterChef
{
  public void membasuh()
  {
     //.step1
  }

  abstract public void masakAyam();
}
```

```
class Chef2 extends MasterChef
{

   public void masakAyam()
   {
      //roaster
   }

}
```

```
class Chef1 extends MasterChef
{

   public void membasuh()
   {
      //step2
   }

   public void masakAyam()
   {
      //stim
   }

}
```

```
class TryRun
{
   psvm()
   {
      MasterChef m1=new MasterChef();
      MasterChef m1=new Chef1();
      Chef1  m2=new Chef1();
      m2.membasuh();  //step2
      Chef2 m3= new Chef2();
      m3.membasuh();   //step1
   }
}
```

interface Vs abstract class in Java.

Source: https://javabypatel.blogspot.com/2017/07/real-time-example-of-abstract-class-and-interface-in-java.html


Germany Kent.
"BE AROUND PEOPLE THAT MAKE YOU WANT TO BE A BETTER PERSON, WHO MAKE YOU FEEL GOOD, MAKE YOU LAUGH, AND REMIND YOU WHAT'S IMPORTANT IN LIFE"
GoFameUS