Course: PARADIGMS AND COMPUTER PROGRAMMING FUNDAMENTALS (PCPF)

Course Instructor

Mrinmoyee Mukherjee B.E (Electronics), M.E (EXTC), PhD (Pursuing)
Assistant Professor
Department of Information Technology
St. Francis Institute of Technology
email: mrinmoyeemukherjee@sfit.ac.in

Academic Year: 2023-24 (Odd Semester)

OUTLINE OF UNIT-2

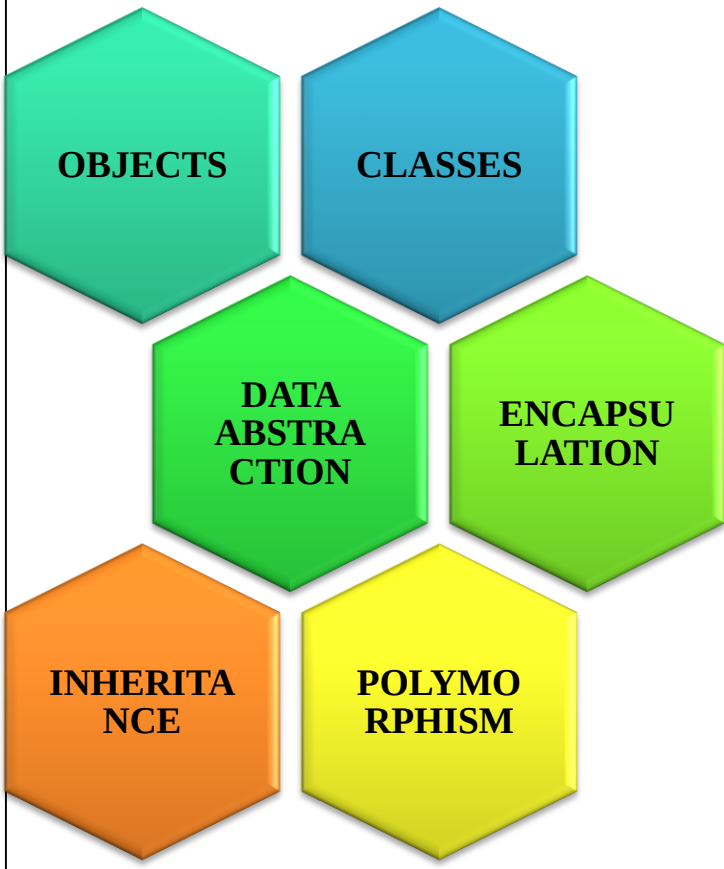
Sub-Unit	Contents
2.1	Grouping of data and operations
2.2	Encapsulation
2.3	Overloading and polymorphism
2.4	Inheritance
2.5	Initialization and finalization
2.6	Dynamic Binding

2.1Grouping of data and operations

FEATURES OF OBJECT-ORIENTED PROGRAMMING

1. Emphasis is on **data** rather than procedure
2. Programs are **divided into objects**
3. Data structures are designed such that they characterize the objects
4. Functions that operate on the data of an object are **tied** together in the data structure
5. **Data is hidden** and cannot be accessed by external functions
6. Objects may **communicate** with each other through functions
7. **New data** and functions can easily be **added** whenever necessary

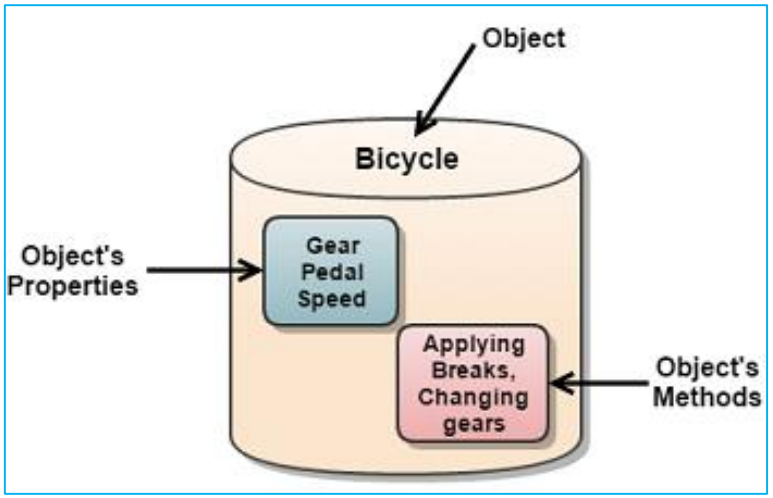
CONCEPTS

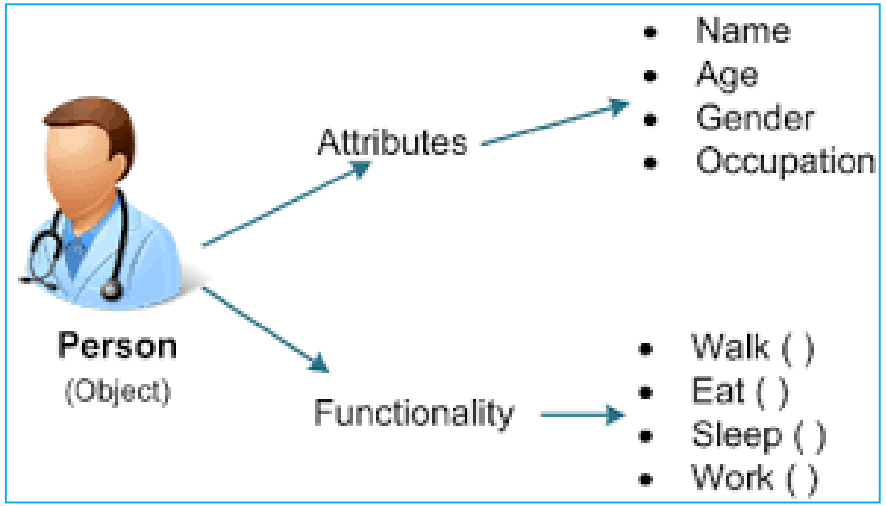


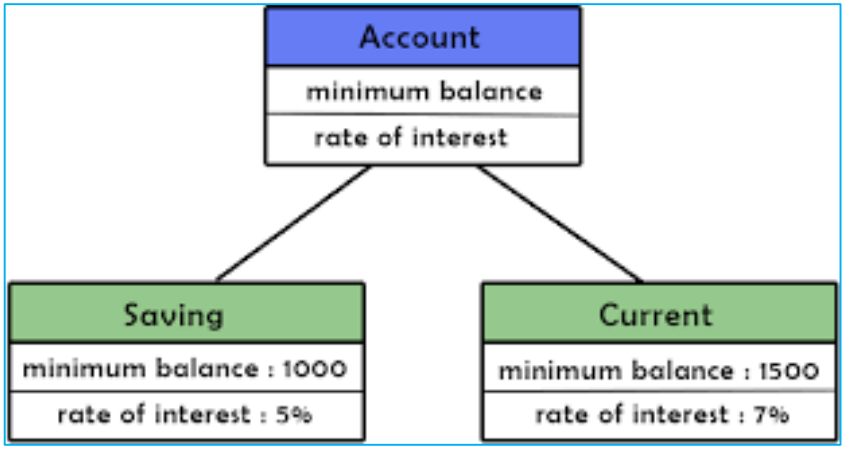
OBJECTS

- Objects are basic run-time entities

- They may represent a person, a place, a bank account, a table of data…..or any other item







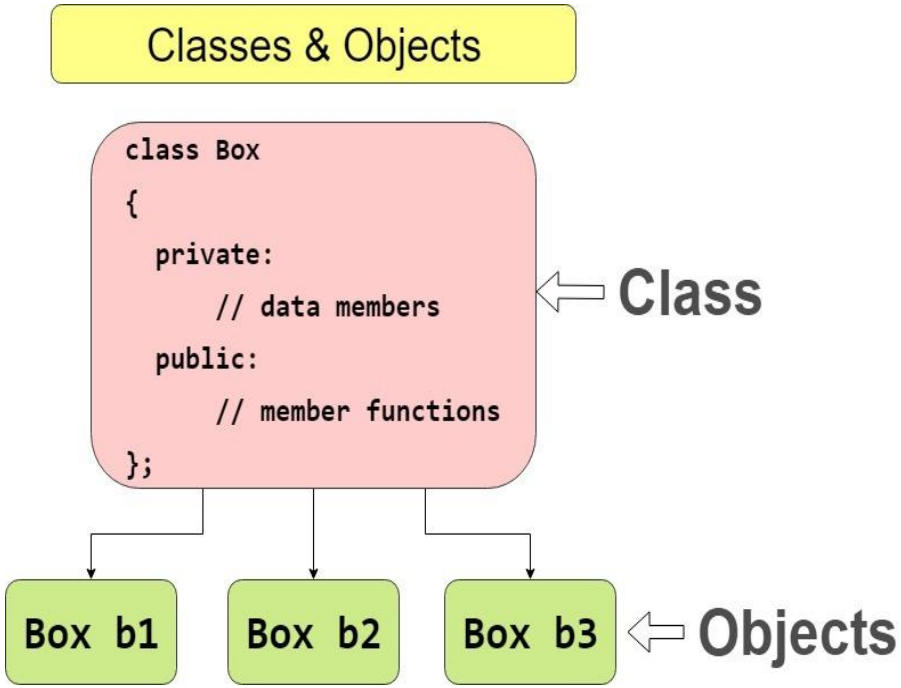
- Objects take up space in the memory and have an associated address

- Each object has an identity, state and behaviour

CLASSES

- Objects contain data and code to manipulate the data
- In-fact objects are variables of type class
- Once a class is created we can create any number of variables of that class
- Each object is associated with the data of type class with which they are created
- A class is thus collection of objects of similar type

Example: mango, apple orange are members of class fruit



One class one object

circledemo.java - Notepad

File Edit Format View Help

```
class circle{
double x,y;
double r;

double circumference(){
return 2*3.141*r;
}

double area(){
return (22/7)*r*r;
}
}

class circledemo{
public static void main(String args[]){

circle c1=new circle();
c1.x=0.0;
c1.y=0.0;
c1.r=5.0;
System.out.println("Circumference is"+c1.circumference());
System.out.println("Circumference is"+c1.area());
}
```

```
C:\Users\m09mu>cd Desktop

C:\Users\m09mu\Desktop>cd Javacodes

C:\Users\m09mu\Desktop\Javacodes>st path="C:\Program Files\Java\jdk1.8.0_251\bin"
'st' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\m09mu\Desktop\Javacodes>set path="C:\Program Files\Java\jdk1.8.0_251\bin"

C:\Users\m09mu\Desktop\Javacodes>javac circledemo.java
circledemo.java:10: error: ';' expected
return (22/7) r*r;
            ^
circledemo.java:10: error: not a statement
return (22/7) r*r;
              ^
2 errors

C:\Users\m09mu\Desktop\Javacodes>javac circledemo.java

C:\Users\m09mu\Desktop\Javacodes>circledemo
'circledemo' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\m09mu\Desktop\Javacodes>java circledemo
Circumference is31.41
Circumference is75.0

C:\Users\m09mu\Desktop\Javacodes>
```

One class more object

circledemo.java - Notepad

File Edit Format View Help

```
class circle{
double x,y;
double r;

double circumference(){
return 2*3.141*r;
}

double area(){
return (22/7)*r*r;
}
}

class circledemo{
public static void main(String args[]){

circle c1=new circle();
circle c2=new circle();
c1.x=0.0;
c1.y=0.0;
c1.r=5.0;
c2.x=0.0;
c2.y=0.0;
c2.r=7;
System.out.println("Circumference is"+" "+c1.circumference());
System.out.println("Circumference is"+" "+c1.area());
System.out.println("Circumference is"+" "+c2.circumference());
System.out.println("Circumference is"+" "+c2.area();
```

```
C:\Users\m09mu\Desktop\Javacodes>javac circledemo.java

C:\Users\m09mu\Desktop\Javacodes>java circledemo
Circumference is31.41
Circumference is75.0
Circumference is43.974000000000004
Circumference is147.0

C:\Users\m09mu\Desktop\Javacodes>javac circledemo.java

C:\Users\m09mu\Desktop\Javacodes>java circledemo
Circumference is 31.41
Circumference is 75.0
Circumference is 43.974000000000004
Circumference is 147.0

C:\Users\m09mu\Desktop\Javacodes>
```

more class more object

```
class circle {
double x,y;
double r;
double circumference(){
return 2*3.1415*r;
}
double area(){
return (22/7)*r*r;
}
}

class box{
double width;
double height;
double depth;

double area(){
double a;
a=(width*height+height*depth+width*depth)*2;
return a;
}
double volume(){
double v;
v=width*height*height;
return v;
}
```

```
class demo1{
public static void main(String args[]){
circle c=new circle();
box b=new box();
//Initialize the circles
c.x=3.0;
c.y=4.0;
c.r=2.0;
b.width=2.0;
b.height=4.0;
b.depth=5.0;
System.out.println("Circumference"+" "+c.circumference());
System.out.println("Area of circle"+" "+c.area());
System.out.println("Area of box"+" "+b.area());
} }
```

Command Prompt

```
1 error

C:\Users\m09mu\Desktop\Javacodes>javac demo1.java
demo1.java:42: error: reached end of file while parsing
} }
   ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac demo1.java

C:\Users\m09mu\Desktop\Javacodes>java demo1
Circumference0.0
Area of circle0.0
Area of box40.0

C:\Users\m09mu\Desktop\Javacodes>javac demo1.java

C:\Users\m09mu\Desktop\Javacodes>java demo1
Circumference 0.0
Area of circle 0.0
Area of box 76.0

C:\Users\m09mu\Desktop\Javacodes>javac demo1.java

C:\Users\m09mu\Desktop\Javacodes>java demo1
Circumference 12.566
Area of circle 12.0
Area of box 76.0

C:\Users\m09mu\Desktop\Javacodes>_
```

2.2:-Encapsulation (C++, Java , Python)

Encapsulation

- The programming technique or process of binding data and corresponding methods together into a single unit called class, (in order to prevent them from being accessed by other classes,) is called data encapsulation.
- If the field is declared private in the class then it cannot be accessed by anyone from outside the class and hides field within the class. Therefore, it is also called data hiding.
- Ex1: School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc. safe.
- Ex2: Suppose you have an account in the bank. Your balance variable should be declared as private for making your account safe.

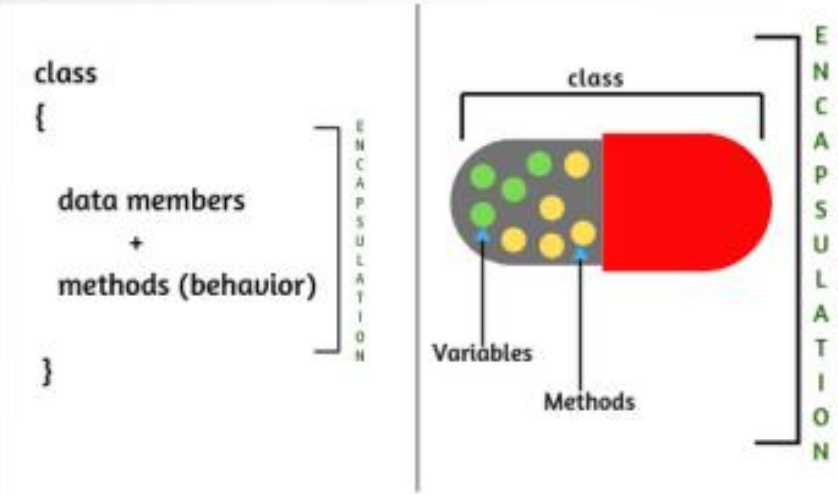


Fig: Encapsulation

- Consider class to create Circle which is 2-d geometrical figure
- Attributes
  - Radius
- Methods
  - Find area
  - Find circumference or perimeter

```
import java.lang.Math;
class Circle {
      double radius = 1.5;

      double getArea(){
              return radius * radius *Math.PI;
              }
       double getPerim(){
              return 2*radius *Math.PI;
              } }

class CreateCircle {
    public static void main(String[] args) {
    Circle c1=new Circle( );
    System.out.println(c1.getArea());
    System.out.println(c1.getPerim());

    Circle c2=new Circle( );
    c2.radius=2.7;
    System.out.println(c2.getArea());
    System.out.println(c2.getPerim());
     } }
```

Class

Object-1

Object-2

Access Modifiers

- Access Modifiers control how much of an object is visible to the rest of Java program.
- Packages provide the highest level of control over object visibility.
- Ex: a class that is not public is not visible outside its own package.
- There are 4 different access modifiers available in java language: **public**, **protected**, **private** and no modifier (**default**)
  - **private** - accessible to the classes only.
  - **protected** - accessible to class, package, and subclasses
  - **default** - accessible to the class and package only
  - **public** - accessible to global

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

PRIVATE ACCESS MODIFIER

- Create class Course
- Attributes
  - courseName
- Methods
  - setCourse
  - getCourse

```
class Course {
  private String courseName = "Python";

  public String getCourse() {
    return courseName;
  }
  public void setCourse(String s) {
    this.courseName = s;
  }
}

class EncapExample {
  public static void main(String[] args) {
    //System.out.println(Course.courseName);
    Course c1 = new Course();
    c1.setCourse("Java");
    System.out.println(c1.getCourse());
}
}
```

Class

Private variable

Error accessing private variable

Object-1

Protected Access Modifier

- **Protected**: This access modifier is used to access the methods or data members of a class within the same package as well as outside the package but only through inheritance.
- The protected access modifier has more accessibility than private and default access modifiers.
- But it has less visibility than the public access modifier.

Creation of Packages

demo.java - Notepad

File Edit Format View Help

```
package p1;

class c1{
public void m1(){
System.out.println("m1 of c1");
}
public static void main(String args[]){
c1 obj = new c1();
obj.m1();
}
}
```

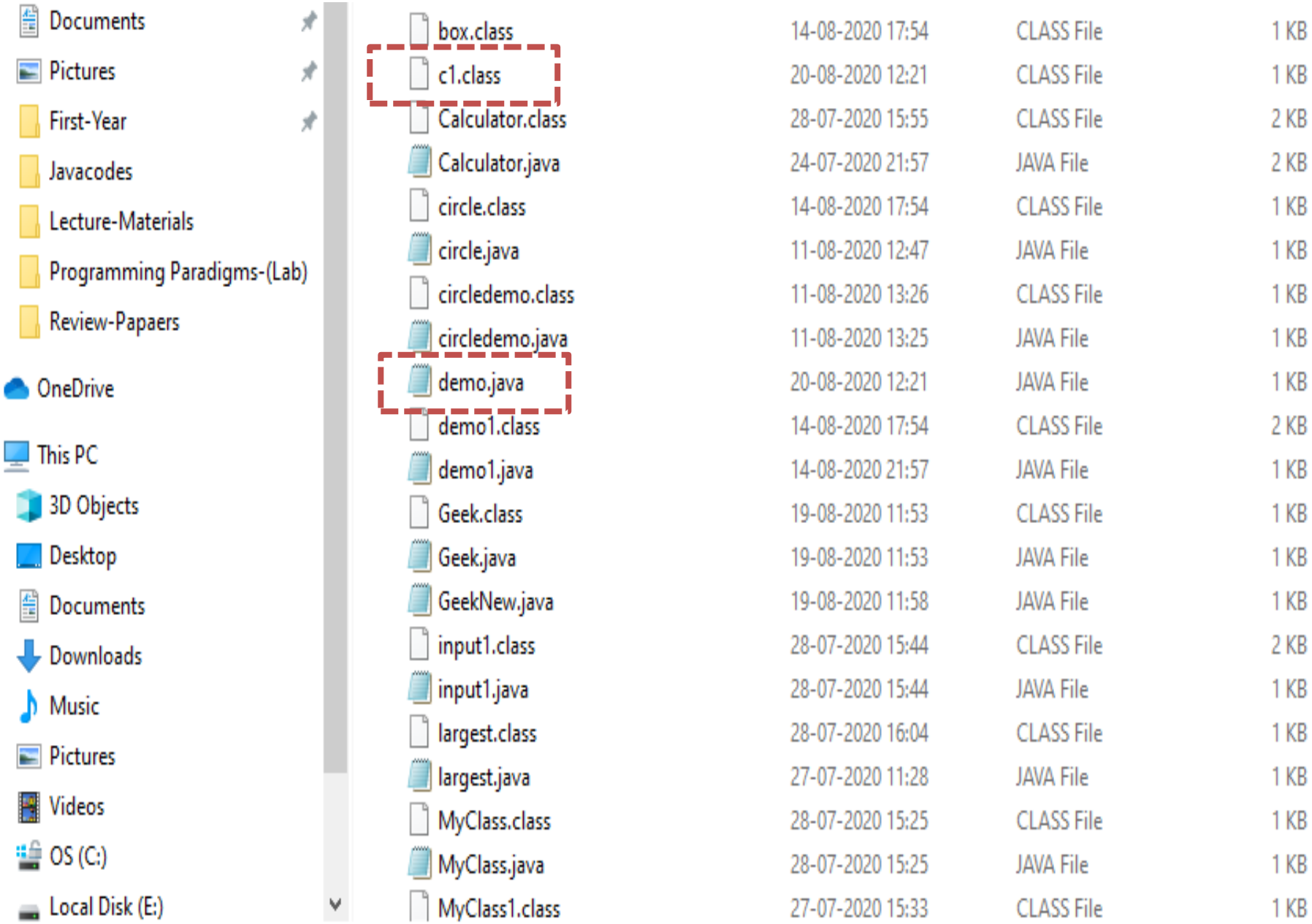
1. Save the file as demo.java

Command Prompt

```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>cd Desktop

C:\Users\m09mu\Desktop>cd Javacodes

C:\Users\m09mu\Desktop\Javacodes>javac demo.java
demo.java:3: error: '{' expected
class c1(){
        ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac demo.java
demo.java:7: error: cannot find symbol
public static void main(string args[]){
                        ^
  symbol:   class string
  location: class c1
1 error

C:\Users\m09mu\Desktop\Javacodes>javac demo.java
```

Compile....In my case I had got few errors

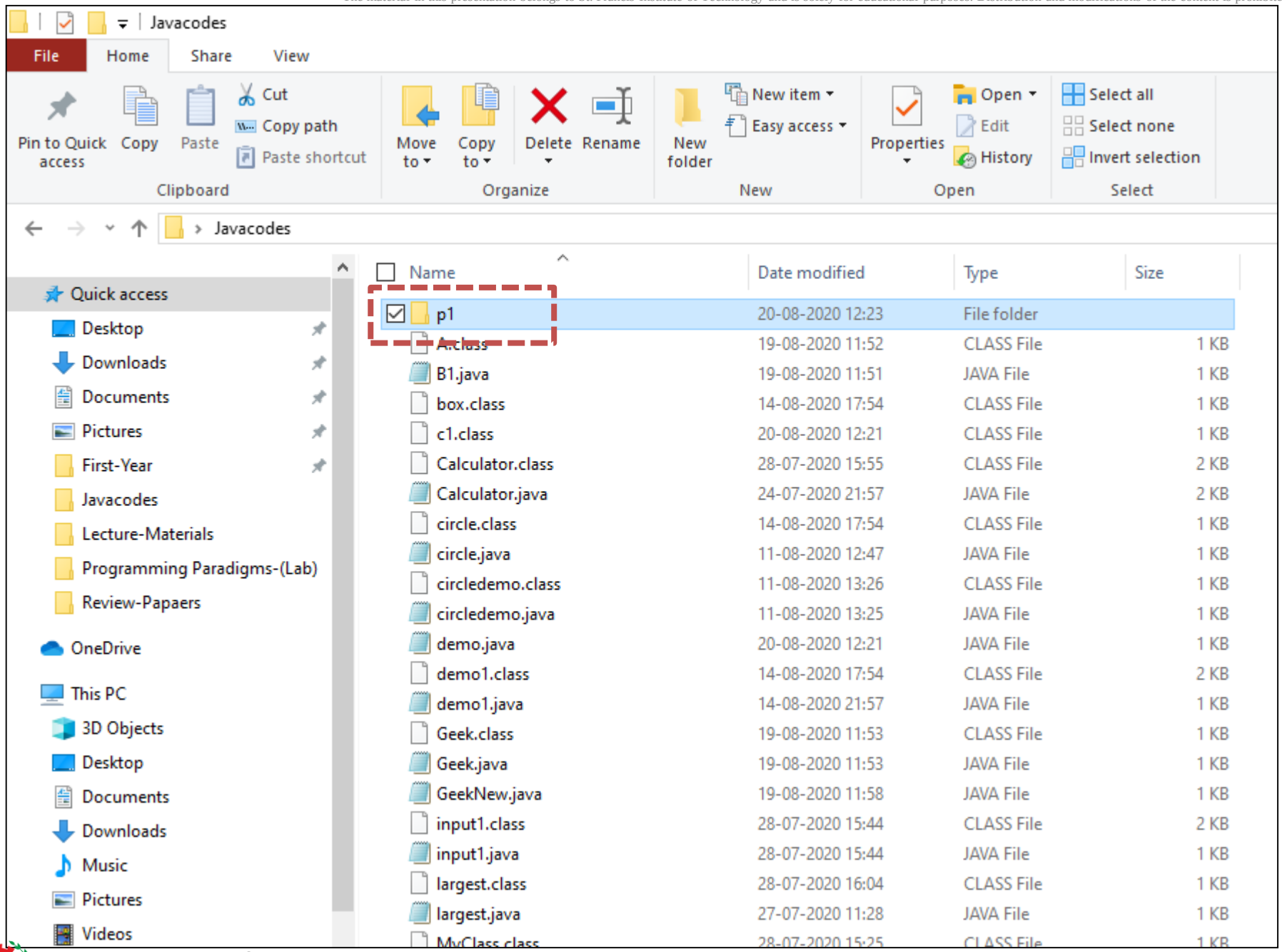
Successful compilation



```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>cd Desktop

C:\Users\m09mu\Desktop>cd Javacodes

C:\Users\m09mu\Desktop\Javacodes>javac demo.java
demo.java:3: error: '{' expected
class c1(){
        ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac demo.java
demo.java:7: error: cannot find symbol
public static void main(string args[]){
                        ^
  symbol:   class string
  location: class c1
1 error

C:\Users\m09mu\Desktop\Javacodes>javac demo.java

C:\Users\m09mu\Desktop\Javacodes>javac -d . demo.java

C:\Users\m09mu\Desktop\Javacodes>javac -d .. demo.java

C:\Users\m09mu\Desktop\Javacodes>_
```

Create a package

Compile file within package



```
C:\Users\m09mu\Desktop\Javacodes>javac demo.java

C:\Users\m09mu\Desktop\Javacodes>javac -d . demo.java

C:\Users\m09mu\Desktop\Javacodes>javac -d .. demo.java

C:\Users\m09mu\Desktop\Javacodes>java p1.c1
m1 of c1

C:\Users\m09mu\Desktop\Javacodes>
```

Run file within package

DATA ABSTRACTION

- Wake up….go to kitchen…put on the coffee machine…..relish the morning cup of coffee

You need to know

- ✓ How to use the coffee machine to make coffee
- ✓ Provide water and coffee beans, switch it on and select the kind of coffee you want to get.

The thing you don't need to know is

- ✓ How the coffee machine is working internally to brew a fresh cup
- ✓ The ideal temperature of the water, amount of ground coffee, sugar added
- **Someone else worried about that and created a coffee machine that now acts as an abstraction and hides all these details**

DATA ABSTRACTION

- Objects in an OOP language provide an abstraction that hides the internal implementation details.

- Similar to the coffee machine in your kitchen, you just need to know

✓ **which methods of the object are available to call and which input parameters are needed to trigger a specific operation**.

- **But you don't need to understand how this method is implemented and which kinds of actions it has to perform to create the expected result**.

- **Definition:** Data abstraction is a process to model/create our own user defined data types (using class and constructs) and then define variables (objects of those new data types)

- Is a simplified view of an object that includes only features one is interested in while hides away the unnecessary details

- Data abstraction becomes an Abstract Data Type (ADT) or user defined type

DATA ABSTRACTION in Java

- Data abstraction in java is achieved through **interfaces and abstract classes**
- The abstract keyword is a non-access modifier, used for classes and methods:
- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).
- An abstract class can have both abstract and regular methods:

```
abstract class Animal  {
 public abstract void animalSound();
public void sleep()
 {
System.out.println("Zzz");
}
 }
```

```
Animal myObj = new Animal(); // will generate an error
```

(will introduce error)

DATA ABSTRACTION in Java-Rules

An abstract class must be declared with an abstract keyword.

Abstract classes can have abstract and non-abstract methods.

It cannot be instantiated. (object cannot be created directly using new keyword)

It can have constructors and static methods also.

It can have final methods which will force the subclass not to change the body of the method.

*myclass2.java - Notepad

File Edit Format View Help

```
//Java Interfaces
//Abstract class

abstract class animal{
//abstract method (does not have a body)
public abstract void animalsound();
//Regular method
public void sleep(){
System.out.println("Hello");
}
}

//Subclass (inherit from animal)
class dog extends animal {
public void animalsound(){
//The body of animal sound is provided here
System.out.println("The dog says hello");
}
}

class myclass2{
public static void main(String[] args){
dog d1=new dog();
d1.animalsound();
d1.sleep();
//animal a1=new animal();
//a1.animalsound();
}
}
```

Declaration of abstract class and methods

Inheritance of abstract class

Creation of object of inherited class

Command Prompt

```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>cd Desktop

C:\Users\m09mu\Desktop>cd Javacodes

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java

C:\Users\m09mu\Desktop\Javacodes>java myclass2
The bog says hello
Hello

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java
myclass2.java:26: error: animal is abstract; cannot be instantiated
animal a1=new animal();
          ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java
myclass2.java:26: error: animal is abstract; cannot be instantiated
animal a1=new animal();
          ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java

C:\Users\m09mu\Desktop\Javacodes>
```

myclass2.java - Notepad

File Edit Format View Help

```
//Java Interfaces
//Abstract class

abstract class animal{
//abstract method (does not have a body)
public abstract void animalsound();
//Regular method
public void sleep(){
System.out.println("Hello");
}
}

//Subclass (inherit from animal)
class dog extends animal {
public void animalsound(){
//The body of animal sound is provided here
System.out.println("The dog says hello");
}
}

class myclass2{
public static void main(String[] args){
dog d1=new dog();
d1.animalsound();
d1.sleep();
//animal a1=new animal();
//a1.animalsound();
}
}
```

- Another way to achieve abstraction in Java, is with interfaces.
- An interface is a completely "**abstract class**" that is used to group related methods with empty bodies:

*Untitled - Notepad

ile Edit Format View Help

```
/ interface
nterface Animal {
 public void animalSound(); // interface method (does not have a body)
 public void run(); // interface method (does not have a body)
```

- To access the interface methods, the interface must be "implemented" by another class with the implements keyword (instead of extends)
- The body of the interface method is provided by the "implement" class:
- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object)
- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final

Command Prompt

```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>cd Desktop

C:\Users\m09mu\Desktop>cd Javacodes

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java

C:\Users\m09mu\Desktop\Javacodes>java myclass2
The bog says hello
Hello

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java
myclass2.java:26: error: animal is abstract; cannot be instantiated
animal a1=new animal();
          ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java
myclass2.java:26: error: animal is abstract; cannot be instantiated
animal a1=new animal();
          ^
1 error

C:\Users\m09mu\Desktop\Javacodes>javac myclass2.java

C:\Users\m09mu\Desktop\Javacodes>javac Dog.java

C:\Users\m09mu\Desktop\Javacodes>java Dog
Interface Method Implemented

C:\Users\m09mu\Desktop\Javacodes>
```

Dog.java - Notepad

File Edit Format View Help

```
interface Pet{
  public void test();
}
class Dog implements Pet{
   public void test(){
     System.out.println("Interface Method Implemented")
  }
   public static void main(String args[]){
     Pet p = new Dog();
     p.test();
  }
}
```

*Untitled - Notepad

File Edit Format View Help

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
```

Declaration of interface and its methods

Extending the interface

Creating object of extended class

*Untitled - Notepad

File Edit Format View Help

```
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```

Declaration of first and second interfaces

Extending the interfaces

Creating object of extended class

- A Java class can implement multiple Java Interfaces. It is necessary that the class must implement all the methods declared in the interfaces.
- Class should override all the abstract methods declared in the interface
- All methods in an interface are implicitly public and abstract
- An interface cannot be instantiated
- An interface which is declared inside another interface is referred as nested interface
- The class cannot implement two interfaces in java that have methods with same name but different return type.

What’s the difference between data abstraction and data encapsulation

DATA ABSTRACTION	DATA ENCAPSULATION
Abstraction is the method of hiding the unwanted information.	Encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
In abstraction, implementation complexities are hidden using abstract classes and interfaces.	encapsulation, the data is hidden using methods

References-

1. Michael L Scott, " Programming Language Pragmatics", Third edition, Elsevier publication (Chapter-9, specifically 9.1 and 9.2)

2. Ravi Sethi, " Programming Languages-concepts and constructs", Pearson Education (Chapter-6)

3. NPTEL lecture series on Programming in Java, IIT Kharagpur
https://www.youtube.com/watch?v=K9gQwLeNXyw&list=PLbRMhDVUMngcx5xHChJ-f7ofxZI4JzuQR&index=8

Course: PARADIGMS AND COMPUTER PROGRAMMING FUNDAMENTALS (PCPF)

Course Instructor

Mrinmoyee Mukherjee B.E (Electronics), M.E (EXTC), PhD (Pursuing)
Assistant Professor
Department of Information Technology
St. Francis Institute of Technology
email: mrinmoyeemukherjee@sfit.ac.in

Academic Year: 2023-24 (Odd Semester)

OUTLINE OF UNIT-2

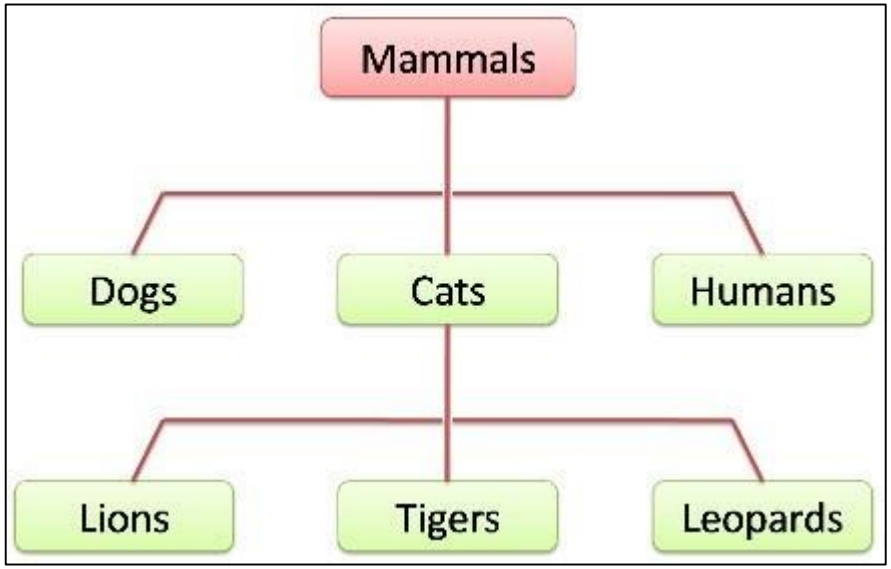
Sub-Unit	Contents
2.1	Grouping of data and operations
2.2	Encapsulation
2.3	Overloading and polymorphism
2.4	Inheritance
2.5	Initialization and finalization
2.6	Dynamic Binding

INHERITANCE

- Inheritance is a relationship between two or more classes where derived class inherits the properties of existing base classes

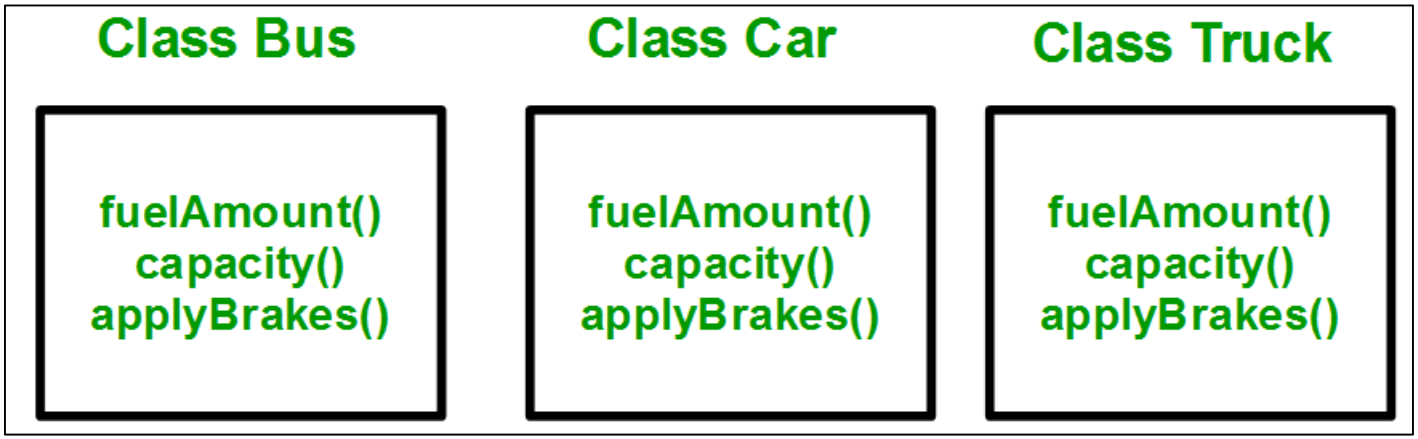
- Base class: It is the class whose properties are inherited by another class. It is also called as Super class or Parent class

- Derived class: It is the class that inherit properties from the base class(es). It is also called sub class or child class

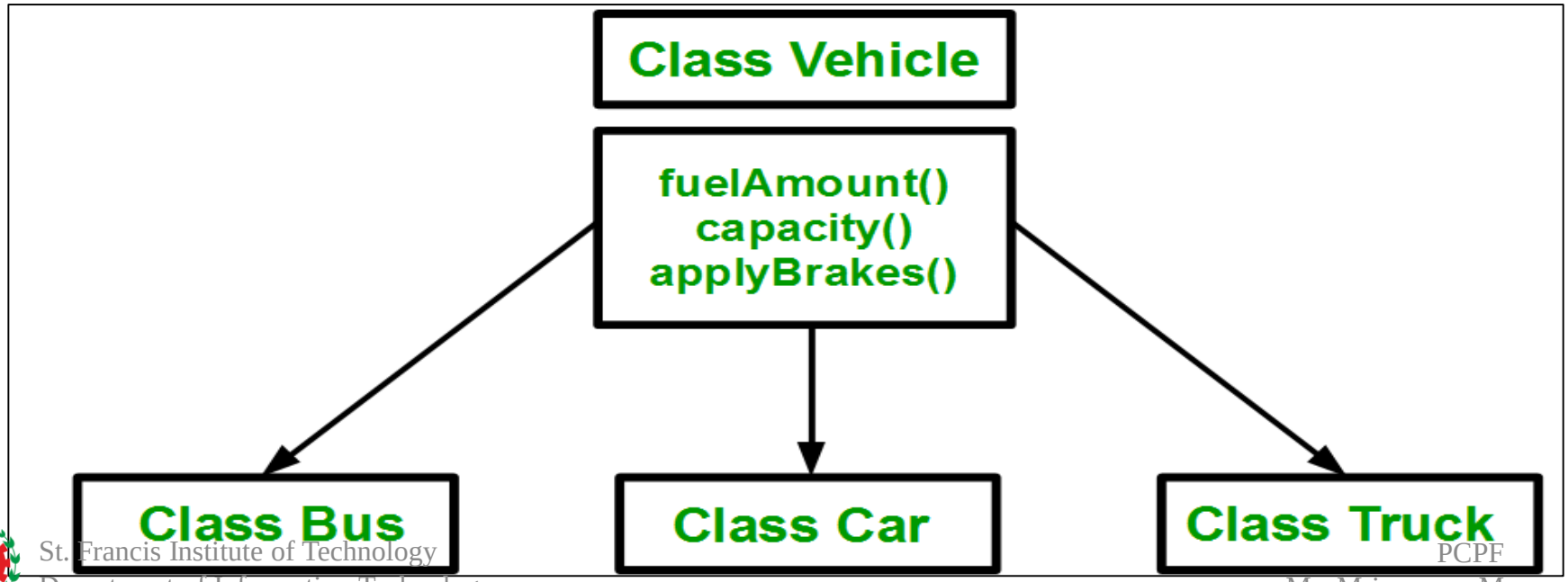


It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class

NEED FOR INHERITANCE



- There is duplication of same code 3 times
- This increases the chances of error and data redundancy.
- To avoid this type of situation, inheritance is used



Example of Inheritance

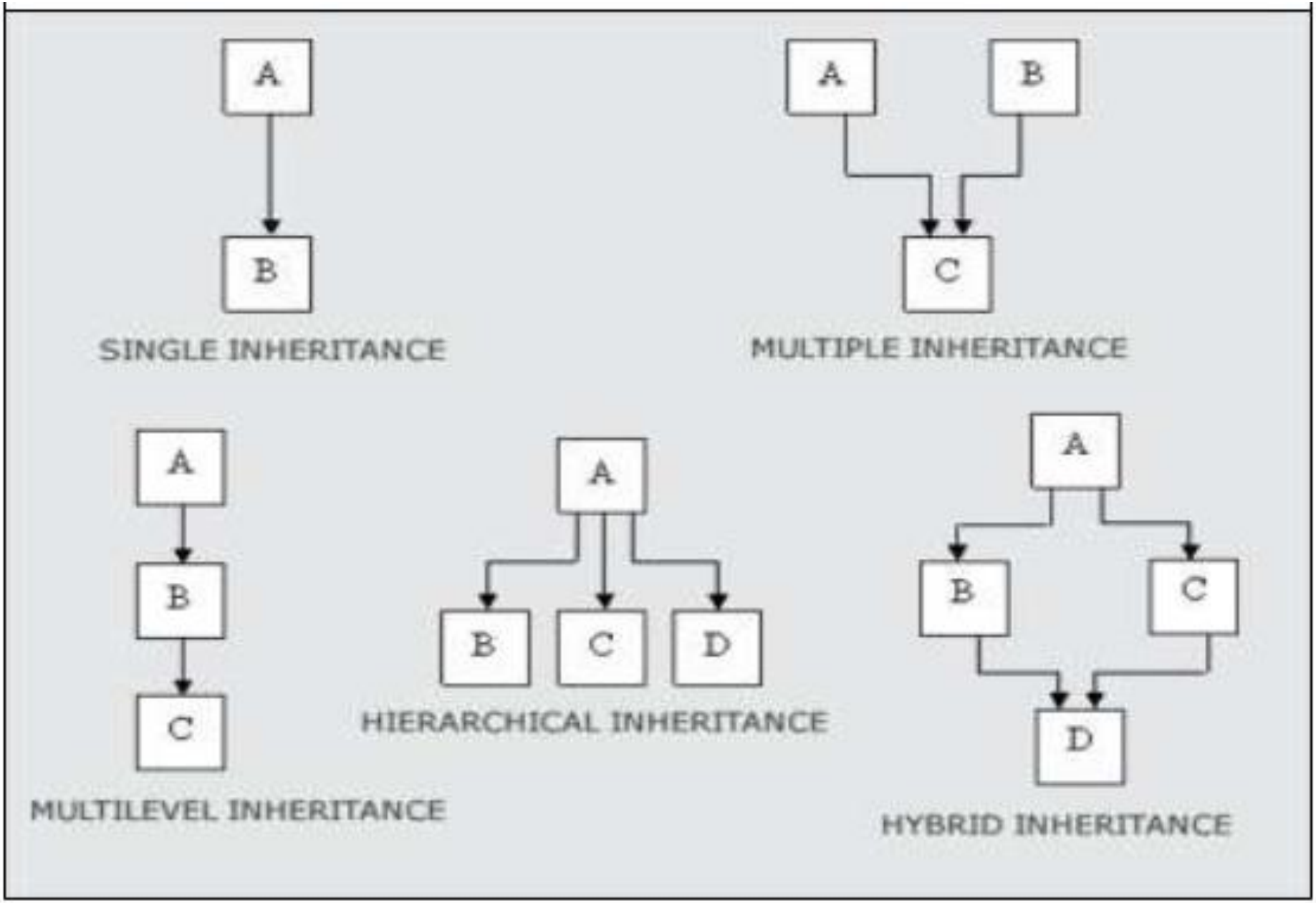
Inheritance in java

```
class Area {
   public int getArea (int l, int b) {
            return l * b;
            }
}
class Rectangle extends Area {
   int length; int breadth;
   public Rectangle() {
        length = 7; breadth = 4;
            }
}

# main class
class myClass1 {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
      Rectangle rt=new Rectangle();
      System.out.println(rt.getArea(4,2));
}}
```

Inheritance in C++

```
#include <iostream>
using namespace std;
class Area {
   public:
   int getArea (int l, int b) {
            return l * b; }
};
class Rectangle : public Area {
    int length; int breadth;
    public:   Rectangle() {
          length = 7; breadth = 4;
                    }
     int area() {
          return Area::getArea(length, breadth);
               }
};
int main()  {
     Rectangle rt;
     cout << "Area : " << rt.area() << end;
     return 0; }
```

TYPES OF INHERITANCE



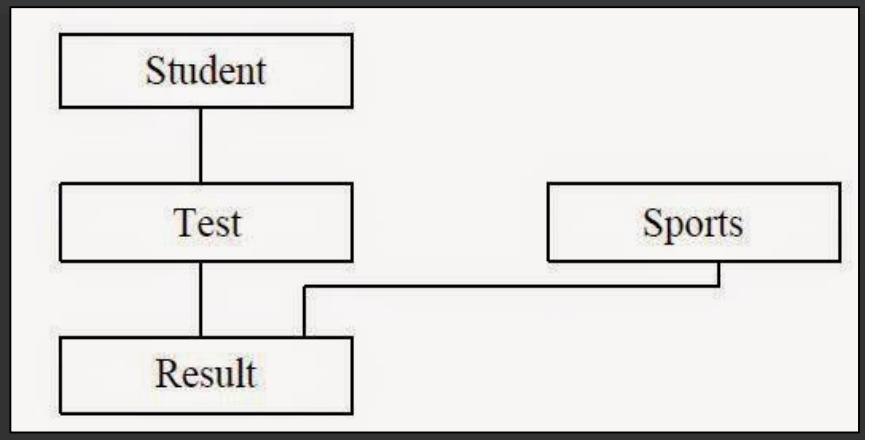
MULTIPLE INHERITANCE

```
#include<iostream>
using namespace std;
//Define class A
class A
{
    protected:
    int m;
    public:
    void get_m(int);
};
void A::get_m(int x)
{
    m=x;
}
//Define class B
class B
{
    protected:
    int n;
    public:
    void get_n(int);
};
void B::get_n(int y)
{
    n=y;
}
```

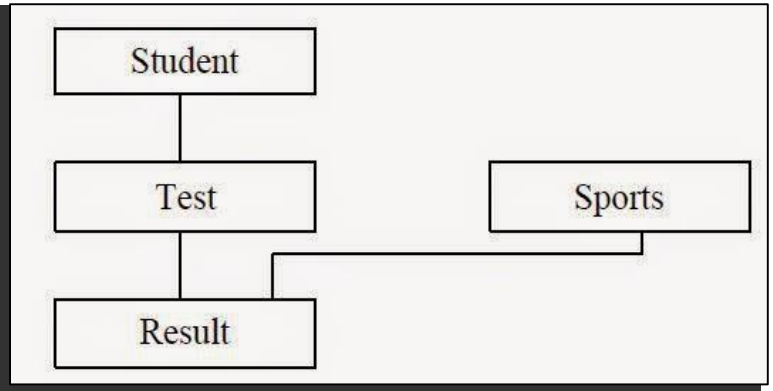
```
//Define class P from A and B
class P:public A,public B
{
    public:
    void display(void);
};
void P::display(void)
{
    cout<<"m="<<m<<endl;
    cout<<"n"<<n<<endl;
    cout<<"m*n"<<m*n<<endl;
}
int main()
{
    P p;
    p.get_m(10);
    p.get_n(20);
    p.display();
    return 20;
}
```

MULTIPLE INHERITANCE

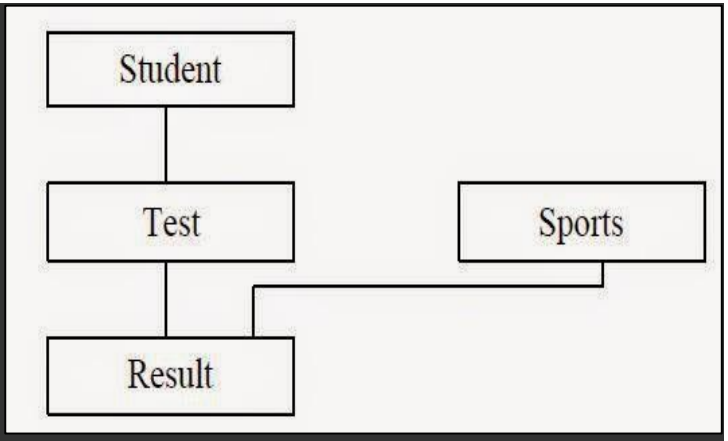

```
#include<iostream>
using namespace std;

class student
{
    protected:
    int rollno;
    public:
    void get_number(int a)
    {
        rollno=a;
    }
    void put_number(void)
    {
        cout<<"The roll number is "<<rollno<<endl;
    }
};
```



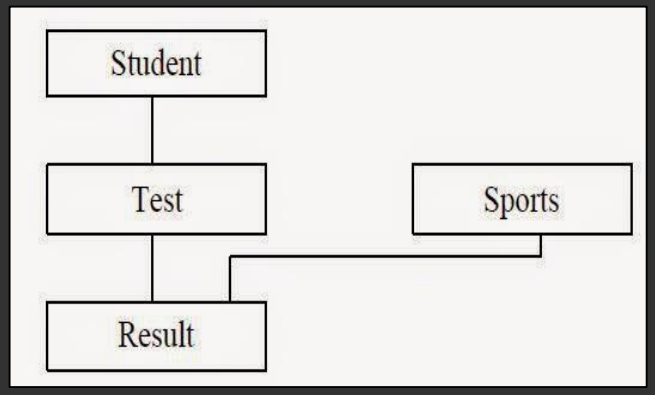
```
class test:public student
{
    protected:
    float p1,p2;
    public:
    void get_marks(float x, float y)
    {
        p1=x;
        p2=y;
    }
    void put_marks(void)
    {
        cout<<"Marks obtained are"<<endl;
        cout<<"Part1 marks are"<<p1<<endl;
        cout<<"Part2 marks are"<<p2<<endl;
    }
};
```



```
class sports
{
    protected:
    float score;
    public:
    void get_score(float s)
    {
        score=s;
    }
    void put_score(void)
    {
        cout<<"The score is "<<score<<endl;
    }
};
```



```
class result: public test,public sports
{
    float total;
    public:
    void display(void)
    {
        total=p1+p2+score;
        put_score();
        put_marks();
        put_number();
        cout<<"Total score"<<total<<endl;
    }
};
```



```
int main()
{
    result r;
    r.get_number(1234);
    r.get_marks(27.5,33.0);
    r.get_score(6.0);
    r.display();
    return 0;
}
```

```
The score is 6
Marks obtained are
Part1 marks are27.5
Part2 marks are33
The roll number is 1234
Total score66.5
```

References-

1. Michael L Scott, " Programming Language Pragmatics", Third edition, Elsevier publication (Chapter-9, specifically 9.1 and 9.2)

2. Ravi Sethi, " Programming Languages-concepts and constructs", Pearson Education (Chapter-6)

3. NPTEL lecture series on Programming in Java, IIT Kharagpur
https://www.youtube.com/watch?v=K9gQwLeNXyw&list=PLbRMhDVUMngcx5xHChJ-f7ofxZI4JzuQR&index=8

Course: PARADIGMS AND COMPUTER PROGRAMMING FUNDAMENTALS (PCPF)

Course Instructor

Mrinmoyee Mukherjee B.E (Electronics), M.E (EXTC), PhD (Pursuing)
Assistant Professor
Department of Information Technology
St. Francis Institute of Technology
email: mrinmoyeemukherjee@sfit.ac.in

Academic Year: 2023-24 (Odd Semester)

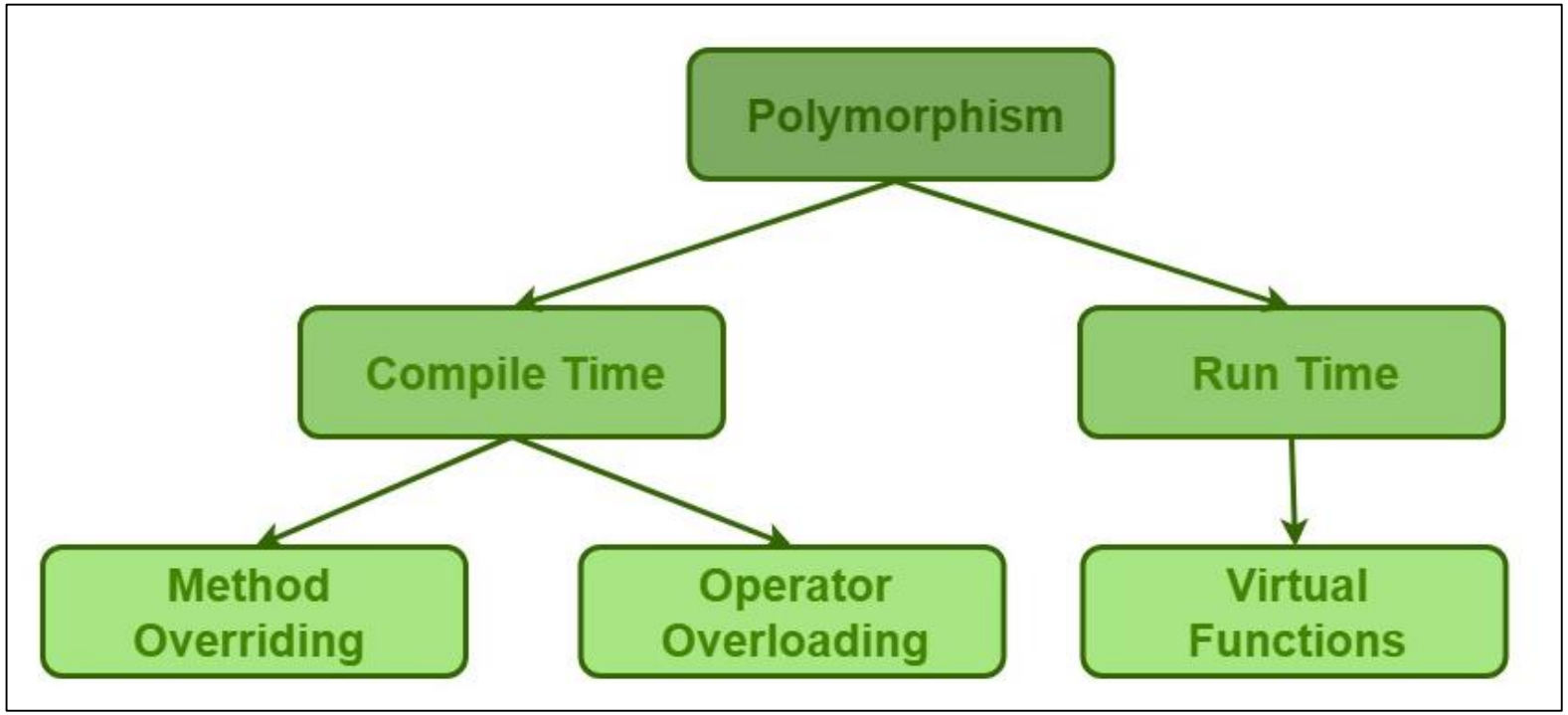
OUTLINE OF UNIT-2

Sub-Unit	Contents
2.1	Grouping of data and operations
2.2	Encapsulation
2.3	Overloading and polymorphism
2.4	Inheritance
2.5	Initialization and finalization
2.6	Dynamic Binding

POLYMORPHISM

- Important concept of object oriented programming
- Polymorphism , a Greek term means the ability to take more than one form
- The behaviour depends upon the types of data used in operation

Eg. *A person possess different behaviour- a father, husband, employee*



Compile Time Polymorphism	Run Time Polymorphism
The function to be invoked is known at the compile time.	The function to be invoked is known at the run time.
It is also known as *overloading*, *early binding and static binding.*	It is also known as *overriding, Dynamic binding and late binding.*
More than one method is having the same name but with the **different** number of parameters or the type of the parameters.	More than one method is having **the same** name, number of parameters and the type of the parameters.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
It provides fast execution as it is known at the compile time.	It provides slow execution as it is known at the run time.
It is less flexible as mainly all the things execute at the compile time.	It is more flexible as all the things execute at the run time.

Compile Time Polymorphism

- Compile-time polymorphism: A type of polymorphism which is achieved by function overloading or operator overloading.
- Function Overloading:
  - When there are multiple functions with the same name but different parameters, then the functions are said to be overloaded. Functions can be overloaded by changing the number of arguments or/and changing the type of arguments. Method overloading increases the readability of the program.
  - There are two ways to overload the method in java
    - By changing number of arguments
    - By changing the data type
- Operator Overloading:
  - C++ also provides the option to overload operators. For example, we can make use of the addition operator (+) for string class to concatenate two strings. We know that the task of this operator is to add two operands. So a single operator '+', when placed between integer operands, adds them and when placed between string operands, concatenates them.
  - Java does not support operator overloading, C++ does

Method Overloading Example

Method overloading: Only one operation, having same name but different parameters

Ex1:

```
class CTimeP {
  void display() {
    System.out.println("Display without
                                parameter");
  }
  void display(String value) {
    System.out.println("In Display with
                        parameter" + value);
  }
}
# main class
public class Main {
  public static void main(String args[]) {
    CTimeP obj1 = new CTimeP();
    obj1.display();
    obj1.display("Polymorphism");
  }}
```

Ex2:

```
class Adder{
   static int add(int a, int b){
            return a+b;
            }
   static int add(int a, int b, int c){
            return a+b+c;
            }
   static int add(double a, double b){
            return a+b;
            }
}
# main class
class TestOverloading{
  public static void main(String[] args) {
    System.out.println(Adder.add(10,12));
    System.out.println(Adder.add(2, 3, 4));
    System.out.println(Adder.add(3.5, 6.4));
            }}
```

Operator Overloading Example

Ex1: Python

```
print(1 + 2)

# concatenate two strings

print("Hi" + "All")

# Product two numbers

print(3 * 4)

# Repeat the String
print("Hello" * 4)
```

Ex2: C++

```
#include<iostream>
using namespace std;
class Complex {
private:  int real, imag;
public:
   Complex(int r = 0, int i =0)  {real = r;   imag = i;}
   Complex operator+ (Complex const &obj) {
      Complex res;
      res.real = real + obj.real;
      res.imag = imag + obj.imag;
      return res;    }
void print() { cout << real << " + i" << imag << endl; }
};
int main()
{
   Complex c1(10, 5), c2(2, 4);
   Complex c3 = c1 + c2; // An example call to "operator+"
   c3.print(); }
```

automatically called when '+' is used with/ betn two Complex objects

Output = 12 + i9

Run Time Polymorphism

- Run-time polymorphism:
- It is also called as Dynamic Method Dispatch
- Whenever an object is bound with the functionality at run time, this is known as runtime polymorphism.
- It is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- Method overriding is an example of runtime polymorphism.
  - In method overriding, a subclass overrides a method with the same signature as that of in its superclass.
  - During compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.
- A virtual function is used to achieve Runtime polymorphism
  - A virtual function is a member function which is declared within a base class using virtual keyword and is re-defined (overridden) by a derived class
  - The resolving of function call is done at runtime.
  - Virtual functions ensure that the correct function is called for an object

Method Overriding Example

Ex1: Java

```
class Animal {
  public void move() {
    System.out.println("Animals can move");
  }}
class Dog extends Animal {
  public void move() {
    System.out.println("Dogs can walk and run");
  }}

public class TestDog {
  public static void main(String args[]) {
    Animal a = new Animal();
    Animal b = new Dog();
    a.move(); // runs the method in Animal class
    b.move(); // runs the method in Dog class
  }
}
```

Ex2: C++

```
#include <iostream>
using namespace std;
class Base {
  public:
   void print() {
      cout << "Base Function" << endl;   }
};
class Derived : public Base {
  public:
   void print() {
      cout << "Derived Function" << endl;   }
};

int main() {
   Derived derived1;
   derived1.print();
   return 0;
}
```

References-

1. Michael L Scott, " Programming Language Pragmatics", Third edition, Elsevier publication (Chapter-9, specifically 9.1 and 9.2)

2. Ravi Sethi, " Programming Languages-concepts and constructs", Pearson Education (Chapter-6)

3. NPTEL lecture series on Programming in Java, IIT Kharagpur
https://www.youtube.com/watch?v=K9gQwLeNXyw&list=PLbRMhDVUMngcx5xHChJ-f7ofxZI4JzuQR&index=8