ITCS214,215,216
Data Structures

(1) Inheritance
(2) Composition
(3) Abstract classes
(4) Interfaces

Last Updated : 24/09/2022

Before we begin
(Review of classes & methods)

Syntax of declaration a class

```
public class className
{
 // Data fields declarations (member variables)

 // constructors

 // methods
}
```

class body

Note that:

- A class is declared by use of **class** Keyword.
- The class body is enclosed between Curly braces { }.
- The class can have only two access modifiers.

  - **public:** class is visible to all classes everywhere.

  - **default (no modifier):** it is visible only within its own package.
- The data or variables defined within a class are called **instance variables**.
- The methods and variables defined within a class are called member of the class.

Data types in Java:

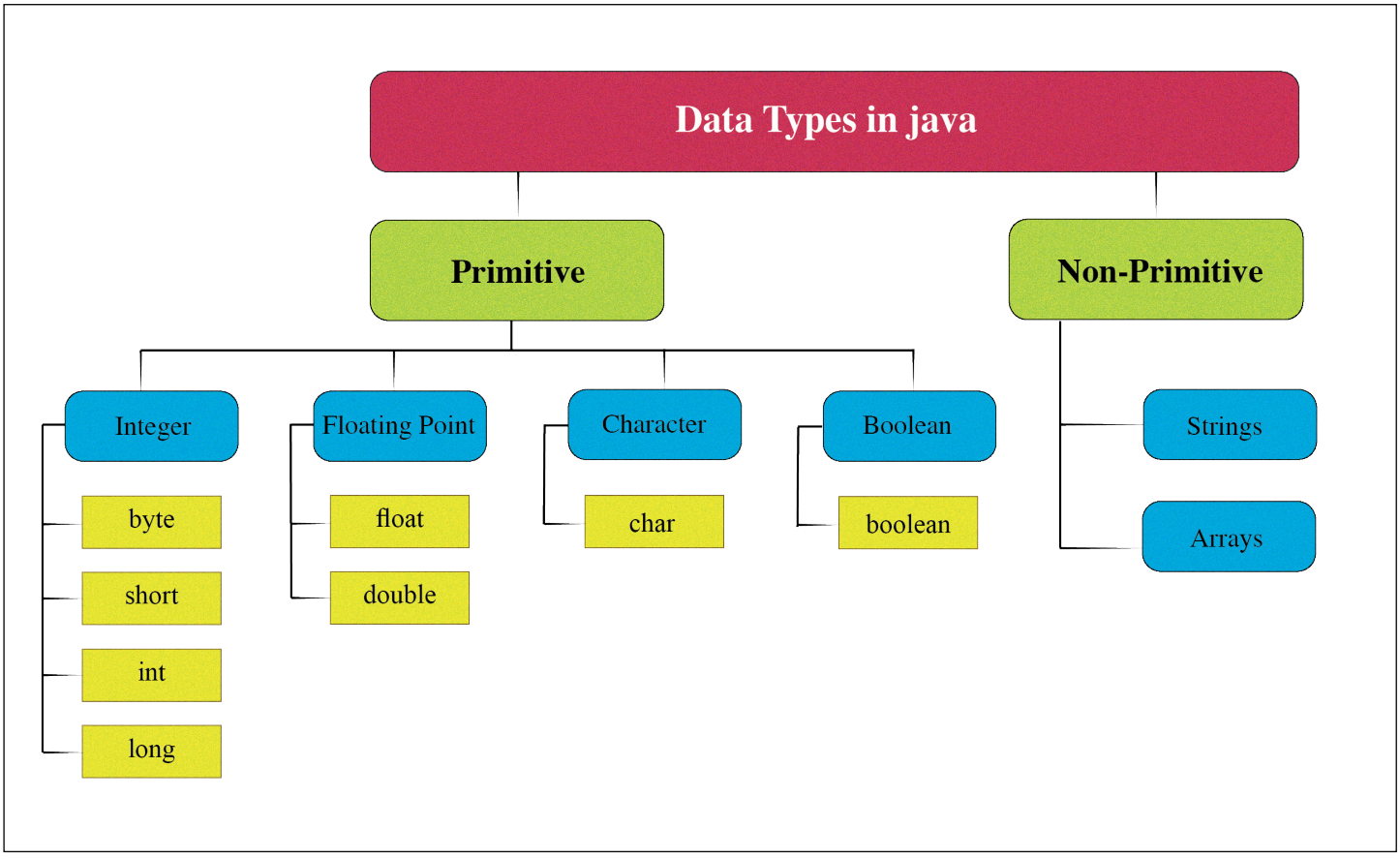
Data types are divided into two groups:

- **Primitive data types:**

  There are 8 primitive data types such as byte, short, int, long, float, double, char and boolean.

- **Non-primitive data types (object data types):**

  such as Strings and Arrays.



All of Java primitive types with amount of computer memory they use (size):

Type Name	Kind of value	Size	Examples
byte	Integer	1 byte	`byte x = 3;`
short	Integer	2 bytes	`short x = 3;`
int	Integer	4 bytes	`int x = 3;`
long	Integer	8 bytes	`long x =3;`
float	Floating Point	4 bytes	`float x =3.5;`
double	Floating Point	8 bytes	`double x = 3.5;`
char	Single Character	2 bytes	`char x ='O';`
boolean		1 bit	`boolean x= false;`

Remember that: some of Escape Characters using with Strings in java	
`\"`	double quote
`\'`	Single quote
`\\`	Backslash
`\n`	New line. Go to the beginning of the next line.
`\t`	Tab. Add whitespace

Declaration of instance variables:

- Instance variables are declared in a class, but outside a method, Constructor or any block.
- An instance Variable Can be declared with these Access modifiers:

  - default (no modifier): accessible only by classes in the same package.

  - public: visible to any class, whether these classes are in the same packages or in another package.

  - private: The member can only be accessed in its own class.

  - protected: The member can be accessed within its only own package.

Examples of declaration a variable

```
long cprNum; //without access modifier

public long cprNum; //with public access modifier

private long cprNum; //with private access modifier

protected long cprNum; //with protected access modifier
```

Static variables (Class variables)

- Static variables are declared with the **static** keyword in a clss (outside a method).
- Static variables are shared by all objects of a class.
- Static variables that are not constants should be private.
- Can be accessed by calling with the class name (doesn't need any object).

```
className.variableName;
```

- Variables declared **static final** are considered constant value (cannot be change).

```
public static final int WEEKDAYS = 7;
```

- We can have a static variables that can change in value, they are declared like instance variables but with the keyword **static**.

```
private static int numberOfVacations;
```

- Both static variables and instance variables are sometimes called **fields** or **data member**.

Remember: Java has three kinds of variables

- Instance variables
- Static variables
- Local variables

Advantages of static variable:

It makes your program memory efficient (It saves memory)

Declaration of methods:

- A method is a block of statements that has a name and can be executed by calling (invoking) it.
- Every program must have at least one method, and every program must have a methed named **main**, which is the method first invoked when the program is run.
- All methods ***-including the main-*** must begin with a method declaration.

Syntax of declaration a method

```
public ReturnType methodName(parameter(s))
{

 // body of the method

}
```

- Variables in a method are called **Local variables**.
- Local variables having the same name and declared in different methods are different variables.

Static methods:

static methods are the methods in java that can be called without creating an object of class.

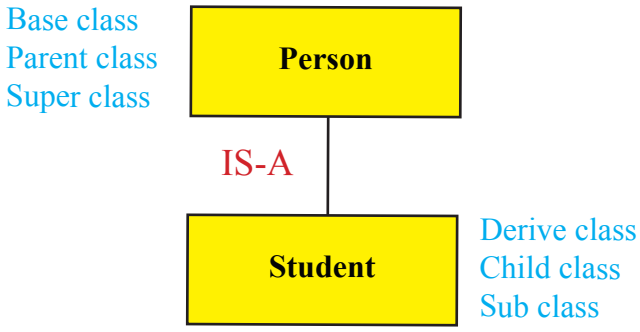
```
ClassName.methodName( );
```

Instance method (non static) & static method

	instance (variables & methods)	static (variables & methods)
instance method	can access directly	can access directly
static method	can't access directly (must use reference to object)	can access directly

(1)
Inheritance

The derived class inherits the attributes and methods from the base class.



IS-A relation is called **inheritance**.

Syntax:

```
public class SuperclassName
{
    // attributes and methods
}


public class SubclassName extends SuperclassName
{
  // attributes and methods
}


public class Test
{
     public static void main (String[]args)
     {
            // define objects
     }
}
```

Example (1):

Write a Super class called **Vehicle** and a child class called **car** as showing in the table below, and then create a main class called **Test** to check all the methods of both classes.

Class	Attributes	Methods
Vehicle (Super class)	• type (String) • color (String) • speed (double)	• setType • gettype • setColor • getColor • setSpeed • getSpeed • print
Car (sub class)	• engineCC(int) • tirePressure(int)	• print

Class Vehicle

```
public class Vehicle
{
    String type;
    String color;
    double speed;

    public void setType(String s) {
        type = s;
    }

    public void SetColor(String c) {
        color = c;
    }

    public void SetSpeed(double sp) {
        speed = sp;
    }


    public String getType() {
        return type;
    }

    public String getcolor() {
        return color;}
```

```
    public double getsSpeed() {
        return speed;}

    public void print() {
        System.out.println("Type = " + type + "\nColor = " + color + "\
nSpeed = " + speed);
    }
} // end of class Vehicle
```

Class Car

```
public class Car extends Vehicle
{
    int engineCC;
    int tirePressure;

    public void print( )
    {
        super.print( );
        System.out.println ( "Engine CC = " + engineCC +
"\nTire Pressure = " + tirePressure);
    }

}  // end of Car class
```

main Class

```
class Test {
    public static void main (String [ ] args )
    {
        Car honda = new Car ();
        honda.setType("pilot");
        honda.SetColor("Blue");
        honda.SetSpeed (250);
        honda.engineCC = 3471;
        honda.tirePressure = 44;
        honda.print();
    }
} // end of main class
```

Output

```
Type = pilot
Color = Blue
Speed = 250.0
Engine CC = 3471
Tire Pressure = 44
```

Example (2):

(A) write a super class called **Person** have the following data members (Private): name(string), cpr(1ong)

and the following methods:

- Default Constructor (without parameters).
- Constructor with two parameters.
- Set and get metheds for name and cpr.
- tostring method to return String equivalent of all attributes.

Class Person

```
public class Person
{
    private String name;
    private long cpr;

    //default Constructor(without parameter)
    public Person  ( )
    {
        this (“unknown” , 0) ;
    }

    // Constructor with Parameter
    public Person(String name, long cpr) {
        this.name = name;
        this.cpr = cpr;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setCpr(long cpr) {
        this.cpr = cpr;
    }

    public long getCpr() {
        return cpr;
    }
```

```
    public String toString ( ) {
       return ("\t\tName : "  + name + "\t\tCPR : " + cpr);
       }

} //end of class Person
```

(B) Write a class **Student** that inherits the properties of class **Person**.This new class will have the following additional data members (private): idNum(String), major (String) and gpa (double)

and the following methods:

- Default Constructor (without Parameters)
- Constructor with 5 Parameters (including that of class Person)
- Set and get metheds for idNum, major and gpa.
- tostring methed to return String equivalent of all attributes .

Class Student

```
public class Student extends Person
{
    private String idNum;
    private String major;
    private double gpa;


//default Constructor
    public Student ( )  {
          this ("Unknown", 0 , " " ," " , 0.0 );
     }

//Constructor with parameters
    public Student(String name, long cpr, String idNum, String major, dou-
ble gpa){
        super(name, cpr);
        this.idNum = idNum;
        this.major = major;
        this.gpa = gpa;}

    public String getIdNum() {
        return idNum;}

    public void setIdNum(String idNum) {
        this.idNum = idNum;}
```

```
    public String getMajor() {
        return major;}

    public void setMajor(String major) {
        this.major = major;}

    public double getGpa() {
        return gpa;}

    public void setGpa(double gpa) {
        this.gpa = gpa;}


    public String toString ( ) {
        return (super.toString( )+ "\t\tIDNumber : " + idNum +
"\t\tMajor : "+ major + "\t\tGPA : "+ gpa);
    }

}// end of class student
```

(C) Write a class Test having only main method. In the main method declare two objects p1 and p2 of type person. create an object of type Person having some suitable values of attributes and assign it to variable p1 .

Also create another object of type Student with suitable valves of its attributes and assign it to p2 then Print attributes of p1 and p2 by Calling toString method.

main Class

```
public class Test
{
    public static void main (String[] args)
    {
        Person p1,p2;
        p1 = new Person( "Sara", 996677818);
        p2 = new Student("Hasan", 010057112, "20193050", "CE", 3.7);
        System.out.println("Person Information :\n" + p1.toString());
        System.out.println();
        System.out.println("Student Information : \n " + p2.toString());
    }
} //end of class Test
```

Output

```
Person Information :
Name : Sara           CPR : 996677818

Student Information :
Name : Hasan     CPR : 2121290    IDNumber : 20193050    Major : CE      GPA : 3.7
```

Questions from previous exams

Question (1):

Assume that class **Item** has following two private data fields:
private String name; private int code;
and following constructor:
public Item(String n, int c) { name = n; code = c; }
Now, we want to write a class called **Electronicltem** that inherits the properties of class **Item** as follows having only one data field:

```
public class Electronicltem extends Item {
        private double price;
```

The constructor of class **Electronicltem** can be written as follows:

(a) public Electronicltem(String n, int c, double p)
{ this(n, c); price = p; }

(b) public Electronicltem(String n, int c, double p)
{ super(n,c); price = p; }

(c) public Electronicltem (String n, int c, double p)
{ Item.super(n, c); price = p; }

(d) public Electronicltem (String n, int c, double p)
{ Item.name = n; Item.code = c; price = p; }

Question (2):

Assume that class **Item** has following two private data fields:
private String name; private int code;
and the following method in addition to other methods:
public String toString( ) { return name + code; }
Now, we want to write a class called **StoreItem** that inherits the properties of class Item as follows, having only one data field:

```
public class StoreItem extends Item {
    private double price;
```

We want to include a method toString in class **StoreItem** to create a String representation of the object, including all attributes, It can be written as follows:
(a) public String toString( ) { return Item.toString() + price; }

(b) public String toString( ) { return super.toString() + price; }

(c) public String toString( ) { return name + code + price; }

(d) Cannot include toString() method in class StoreItem as there is already a toString() method in class Item.

Exercise:

Consider the following class definition:

```
public class Person
{
    private String name;
    private long cprNum;
    public Person( ){
        this("Unknown", 0);
    }
    public Person(String pName, long code){
        setName(pName);
        setCprNum(code);
    }
    public void setName(String pName){
        name = pName;
    }
    public boolean setCprNum(long code){
        if (code > 0 && code < 1000000000){
            cprNum = code;
            return true;
        }
```

```
        else{
            System.out.println("Invalid CPR, initializing it to 0");
            cprNum = 0;
            return false;
        }
    }
    public String getName(  ){
        return name;
    }
    public long getCprNum( ){
        return cprNum;
    }
    public String toString( ){
        return("Name:  " +  name + "CPR:  " +  cprNum);
    }
} // end of class Person
```

(A) Write a class called **Employee**, which inherits the properties of class **Person**. This new class has the following additional data fields (private): position (String), salary (double)

Methods:

- default constructor (without parameters).
- constructor with 4 parameters: name, cpr, position and salary.
- set and get methods for both data members.
- toString method to return String equivalent of all attributes (including that of Person).

(B) Write a class **Test** having only main method. In main method declare two objects **p1** and **p2** of type **Person**. Create an object of type **Person** having some suitable values of attributes and assign it to variable **p1**. Also create another object of type Employee with suitable values of its attributes and assign it to **p2**. Now, print attributes of **p1** and **p2** by calling toString methods.

(2)
Composition

The Composition is a way to design or implement the **part-of** relationship. In Composition we use an instance variable that refers to another object.

We can say it is a technique through which we can describe the reference between two or more classes. And for that, we use the instance variable, which should be created before it is used.

Example (1):

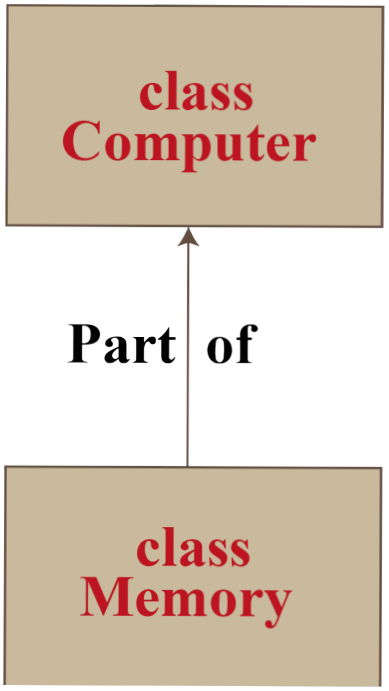
(A) Write a class called **Memory** having the following members:

Data members (private):

- type of type String,
- size of type int,
- speed of type int.

Methods (public):

- set methods for all three data members,
- print method to print all attributes.




```
public class Memory
{
    private String type;
    private int size;
    private int speed;

    //set methods
    public void setType(String t) {type=t; }
    public void setSize(int s) {size = s ;}
    public void setSpeed(int s) { speed = s ; }


    public void print ( ) {
    System.out.println ( "The memory Specifications:" ) ;
    System.out.println  ( "Type : " +type+"\nSize : " + size +"\nSpeed : "
+ speed);
    }

} //end of class Memory
```

(B) Write a class called **Computer** having the following members:

Data members (private):

- processor of type String.
- mem of type Memory.

Methods (public):

- setComputer to set values for the processor and mem.
- print method to print all attributes (including of class Memory).

```
public class Computer
{
    private String processor;
    private Memory mem; //Composition

    public void setComputer (String p, String t, int s, int sp) {
    processor = p ;
    mem = new Memory( ) ;
    mem.setType (t);
    mem.setSize (s);
    mem.setSpeed (sp);
    }

    public void print( )
    {
        System.out.println ("processor :" + processor) ;
        mem.print( ) ;
    }
} //end of class Computer
```

(C) Write a Java application called **ComputerTest** having only main method to test your program. Create an object pc of type Computer having following values:
processor = Intel , type = DDR4, size = 8, speed = 1333

```
public class ComputerTest
{
    public static void main ( String [ ] args )
{
      Computer pc = new Computer();
      pc.setComputer("Intel", "DDR4", 8, 1333);
      pc.print();

    }
} // end of class ComputerTest
```

Output

```
processor :Intel
The memory Specifications:
Type : DDR4
Size : 8
Speed : 1333
```

Example (2):

Consider the following class definition:

```
public class AddressType
{
    private int buildingNum;   //Building number
    private int roadNum;       // Road number
    private int blockNum;      // Block number
    private String area;

// constructor
     public AddressType(int bu, int ro, int bl, String ar){
        buildingNum = bu;
        roadNum = ro;
        blockNum = bl;
        area = ar;}

    public void setAddress(int bu, int ro, int bl, String ar)
    {
        buildingNum = bu;
        roadNum = ro;
        blockNum = bl;
        area = ar;
    }

    public int getBuildingNum( ) { return buildingNum; }
    public int getRoadNum( ) { return roadNum; }
    public int getBlockNum( ) { return blockNum; }
    public String getArea( ) { return area; }

    public void print( )
    {
        System.out.println("Building Number = " + buildingNum +
                "Road Number = " + roadNum + "Block Number = " + blockNum +
                "Area = " + area);
    }

} // end of class AddressType
```

Write a class called **Building**, having the following members:

Data members (private):

- address of type AddressType,
- floorArea of type float,
- numOfFloors of type int.

Methods (public):

- constructor with 6 parameters.
- set and get methods for all three data members,
- print method to print all attributes (including address),

```
public class Building {
    private AddressType address;
    private float floorArea;
    private int numOfFloors;

    public Building(int bu,int ro,int bl,String ar, float floorArea, int nu-
mOfFloors) {
        address = new AddressType(bu,ro,bl,ar);
        this.floorArea = floorArea;
        this.numOfFloors = numOfFloors;
    }

    public AddressType getAddress() {
        return address;
    }

    public void setAddress(AddressType address) {
        this.address = address;
    }

    public float getFloorArea() {
        return floorArea;
    }

    public void setFloorArea(float floorArea) {
        this.floorArea = floorArea;
    }

    public int getNumOfFloors() {
        return numOfFloors;
    }

    public void setNumOfFloors(int numOfFloors) {
        this.numOfFloors = numOfFloors;
    }

    public void print (){
        address.print();
        System.out.println("Floor Area: " + floorArea);
        System.out.println("Num Of Floors " + numOfFloors);
    }

}//end of class Building
```

Questions from previous exams

Question (1):

Assume that class **Person** has following two private data fields:
private String name; private int cpr;

and following constructor:
public Person(String n, int c) { name = n; cpr = c; }
Now, we want to write a class called **Student** having three data fields:

```
public class Student {
        private Person person; private int id;  private int major;
```

The constructor of class **Student** can be written as follows:

(a) public Student(String n, int c, int d, int m)
{ name = n; cpr = c; id = d; major = m; }

(b) public Student(String n, int c, int d, int m)
{ person.name = n; person.cpr = c; id = d; major = m; }

(c) public Student(String n, int c, int d, int m)
{ person = new Person(n, c); id = d; major = m; }

(d) public Student(String n, int c, int d, int m)
{ super.name = n; super.cpr = c; id = d; major = m; }

Question (2):

Assume you are given a class called **Tyre** that has two attributes **type** and **size** of type String. There is another class called **Car** that has three attributes **name** of type String, **num** of type integer and **t** of type Tyre, the constructor of class **Car** can be written as:

(a)
```
public Car(String t, String s)
    { name = "Unknown";
      num =  "Unknown";
      type = t;
      size = s;}
```

(b)
```
public Car(String n, int no, String t1, String s)
    { name = n;
      num =  no;
      t = new Tyre (t1,s);}
```

(c)
```
public Car(String n, int no, Tyre tyre)
    { name = n;
      num =  no;
      t.type = tyre.type;
      t.num = tyre.num;}
```

(d)
```
public Car(String n, int no)
    { name = n;
      num =  no;
      type = "Unknown";
      size = "Unassigned";}
```

(3)
Abstract Classes

As abstract class will only allow me to create an object from the derived class.

Syntax of Abstract class:

```
public abstract class  className            //OR abstract public
{
 // data field declarations
 // actual methods
 // abstract methods

public abstract dataType methodName (Parameter List);

}
```

Notes:

- Abstract class in java can't be instantiated.
- Use **abstract** keyword to create an abstract method.
- An abstract method doesn't have body.
- If a class have abstract methods, ***then the class should also be abstract or interface.***
- It's not necessary for an abstract class to have an abstract method.

Example (1):

(A) Write an abstract class called **BankAccount** has the following private data fields: accountNum(long), balance(double), name(String)

and the following methods:

1. Constructor having 3 parameters for all data fields.
2. Get methods for all three data fields.
3. Method deposit to **deposit** money in the account and update balance.
4. Abstract method **withdraw** to withdraw money from the account and update balance.
5. Abstract method **updateBalance** to update balance according to parameter.
6. **toString** method to return String equivalent of all data fields.

BankAccount Class

```
public abstract class BankAccount
{
    private long accountNum;
    private double balance;
    private String name;

    public BankAccount(long accountNum, double balance, String name) {
        this.accountNum = accountNum;
        this.balance = balance;
        this.name = name;
    }

    public long getAccountNum() {
        return accountNum;
    }

    public double getBalance() {
        return balance;
    }

    public String getName() {
        return name;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public void deposit (double amount)
    {
        balance += amount;
    }

    public abstract void withdraw (double amount);

    public abstract void updateBalance (double amount);

    public String toString()
    {
        return ("Account Number : " + accountNum + "\t\tBalance : " +
balance + "\t\tName : " + name);
    }

} //end of class BankAccount
```

B) Write a class called **SavingAccount** that inherits the properties of abstract class **BankAccount** and having the following additional data fields: interestRate(double)
and the following methods:

1. Constructor having 4 Parameters for all data fields including that of class BankAccount.
2. set and get methods for interestRate.
3. Implementation of method withdraw.
4. Implementation of method updateBalance.
5. toString method to return String equivalent of all data fields.

SavingAccount Class

```
public class SavingAccount extends BankAccount
{
    private double interestRate;

    public SavingAccount(long accountNum, double balance, String name, dou-
ble interestRate) {
        super(accountNum, balance, name);
        this.interestRate = interestRate;
    }

    public double getInterestRate() {
        return interestRate;
    }

    public void setInterestRate(double interestRate) {
        this.interestRate = interestRate;
    }

    public void withdraw (double amount)
    {
        if (amount <= getBalance())
            setBalance(getBalance()-amount);

        else
            System.out.println("Sorry, you don't have enough balance");
    }

    public void updateBalance (double amount)
    {
        setBalance(amount);
    }
```

```
    public String toString()
    {
        return (super.toString() + "\t\t Interest Rate : " + interestRate);
    }

}//end of class SavingAccount
```

C) Write a class called **Test** having only main method to test all functionalities of the methods.

Test Class

```
public class Test
{
    public static void main(String[] args) {
        SavingAccount c1 = new SavingAccount(200112,200,"Marim",5);
        c1.updateBalance(500);
        System.out.println(c1.toString());
        c1.withdraw(50);
        c1.deposit(530);
        System.out.println(c1.toString());
        c1.withdraw(1000);

    }
}//end of class Test
```

Output

```
Account Number : 200112    Balance : 500.0    Name : Marim    Interest Rate : 5.0
Account Number : 200112    Balance : 980.0    Name : Marim    Interest Rate : 5.0
Sorry, you don't have enough balance
```

Questions from previous exams

Question (1):

Assume **Furniture** is an abstract class having an abstract method called **calculatePrice**, The method can be written correctly within the **Furniture** class as follows: (Note that {.....} means there is some code inside the curly parenthesis).

(A) public abstract double calculatePrice();

(B) public double abstract calculatePrice();

(C) public double calculatePrice() {.....}

(D) public double abstract calculatePrice() {.....}

Question (2):

Assume are given one **abstract** super class called **Forest**, and you have a sub-class (derived class) called **Jungle** that inherits the properties of class **Forest**. Abstract method called **nature()** is declared in class **Forest** and class **Jungle** has an implementation of the abstract method **nature()** given in the **Forest** abstract class.

Which of the following is correct implementation of method method nature().

(A) public abstract void nature() {.....}

(B) public void abstract nature () {.....}

(C) public void nature () {.....}

(D) all are correct.

Question (3):

Consider the following class definition:

```
public abstract class Employee
{
    private String name;
    private long cprNum;
    private String position;
    public Employee( ){
        this("Unknown", 0, "Unknown");
    }
    public Employee(String eName, long code, String pos){
        setName(eName);
        setCprNum(code);
        setPosition(pos);
    }
    public void setName(String eName){
        name = eName;
    }
    public boolean setCprNum(long code){
        if (code > 0 && code < 1000000000){
            cprNum = code;
            return true;        }
        else{
            System.out.println("Invalid CPR, initializing it to 0");
            cprNum = 0;
            return false;
        }
    }
    public void setPosition(String pos){
        position = pos;}

    public String getName(  ){
        return name;}

    public long getCprNum( ){
        return cprNum;}

    public String getPosition(  ){
        return position;}

    public abstract double salary( ); // abstract method

    public String toString( ){
        return("Name: " + name + "\tCPR: " + cprNum + "\tPosition: " +
                position);
    }
} // end of class Employee
```

Write a class called **FullTimeEmployee**, which inherits the properties of class **Employee**. This new class has the following additional data fields (private):
basicSalary(double), allowances(double), deductions(double).

and the following methods:

- Default constructor (without parameters)
- Constructor with 6 parameters for name, cpr, position, basicSalary, allowances, and deductions.
- set and get methods for all three data members separately.
- Implementation of abstract method salary that returns the salary as:

basicSalary + allowances – deductions

- toString method to return String equivalent of all attributes (including that of Employee).

```
public class FullTimeEmployee extends Employee {
    private double basicSalary;
    private double allowances;
    private double deduction;

    public FullTimeEmployee() {
        this("Unknown", 0, "Unknown", 0, 0, 0);
    }


public FullTimeEmployee(String eName, long code, String pos, double b,
                        double a, double d) {
        super(eName, code, pos);
        setBasic(b);
        setAllowance(a);
        setDeduction(d);
    }

    public void setBasic(double b) {
        basicSalary = b;}

    public void setAllowance(double a) {
        allowances = a;}

    public void setDeduction(double d) {
        deduction = d;}

    public double getBasic() {
        return basicSalary;}
```

```
     public double getAllowance() {
        return allowances;}

    public double getDeduction() {
        return deduction;}

    public double salary() {
        return (basicSalary + allowances - deduction);}

// for including the net salary is not in the question
    public String toString() {
        return (super.toString() + "\tBasic Salary: " + basicSalary + "\
tAllowances: " + allowances + "\tdeduction: " + deduction + "\tNet Salary:
" + salary());
    }

}
```

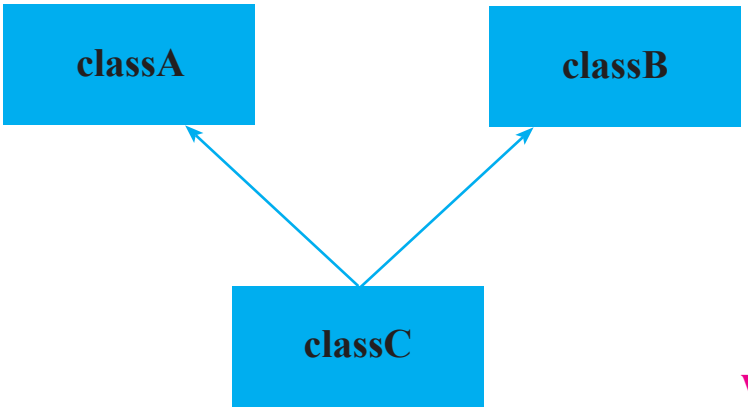
(4)
Interfaces

It is a collection of abstract methods with empty bodies, An interface is different from a class in Several ways:

- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields, The only fields that can appear in interface must be declared both static and final.
- An interface is not extend by a class, it is **implemented** by a class.
- The class can implemant multiple interface.

Syntax of interface:

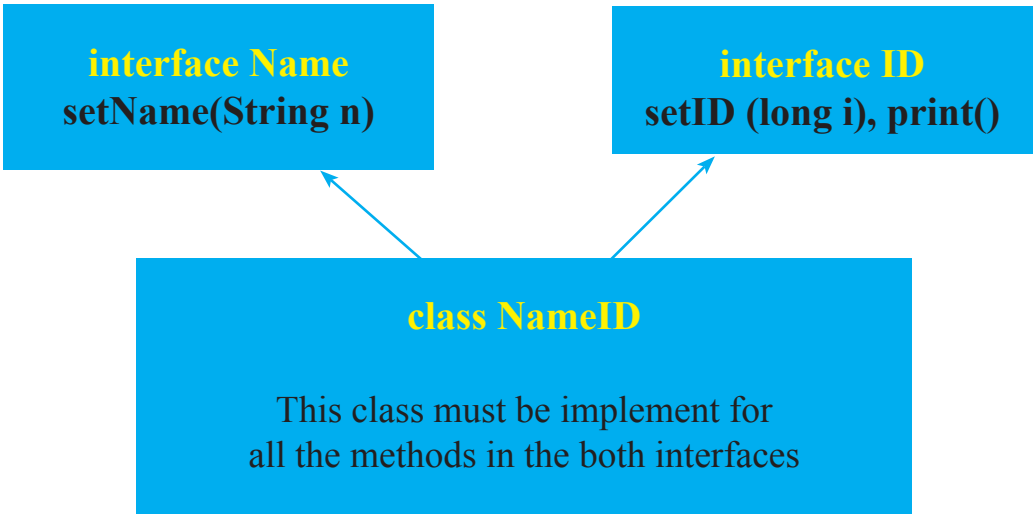
```
public interface interfaceName
{
 // constant declarations
 // abstract methed declarations
}
```



wrong, is not valid in java
we can use inerface

public class **classA** { ... }
public class **classB** { ... }
public class **classC** extends classA , classB { ... }

Example (1):



(A) Write an interface called **Name** having an abstract method **SetName**.

```
public interface Name
{
    public abstract void setName(String n);
}
```

(B) Write an interface called **ID** having an abstract methods **setID and print**.

```
public interface ID
{
    public abstract void setId(long i);
    public abstract void print( );
}
```

(C) Write a class **NameId** that implements all the abstract methods of the both interfaces defined in part (A) and (B), The class having the following data fields (private):
name (String), id (long).

```
public class NameId implements Name,ID
{
    private String name ;
    private long id ;

    public void setName(String n) { name = n;}
    public void setId(long i) { id = i; }

    public void print( ){
        System.out.println ("Name: " + name +"\t\tID: " + id); }

} //end of class NameId
```

(D) Write a Java application called **Check** having only main method to test your class NameId. Create an object **x** of type **NameID** having the following values of data fields:
name = Yousif,
id = 991054301

```
public class Check {

    public static void main (String[ ]args)
    {
        NameId x = new NameId ();

        x.setName("Yousif");
        x.setId(991054301);
        x.print();
    }
} //end of class Check
```

Output

```
Name: Yousif      ID: 991054301
```

Example (2):

(A) Write an interface called **Vehicle** having following abstract methods:

- getSittingCapacity: to return sitting capacity,
- getManufacturer: to return the manufacturer
- getPrice: to return price
- age: to return 2020 – yearModel // age of vehicle
- getColor: return color
- toString: to return String representation of all data fields.

```
public interface Vehicle {
    public abstract int getSittingCapacity ();

    public abstract String getManufacturer ();

    public abstract double getPrice ();

    public abstract int age ();

    public abstract String getColor ();

    public abstract String toString ();

}//end of inteface Vehicle
```

(B) Write a class **Car** that implements all the abstract methods of the interface **Vehicle** defined in part (A) and having the following data fields:
sittingCapacity (int), manufacturer (String), price (double), yearModel (int), color (String).

```
public class Car implements Vehicle {
    private int sittingCapacity;
    private String manufacturer;
    private double price;
    private int yearModel;
    private String color;

    public Car(int sittingCapacity, String manufacturer, double price, int
yearModel, String color) {
        this.sittingCapacity = sittingCapacity;
        this.manufacturer = manufacturer;
        this.price = price;
        this.yearModel = yearModel;
        this.color = color;
    }

    public int getSittingCapacity ()
    {
        return sittingCapacity;
    }

    public String getManufacturer (){
        return manufacturer;
    }

    public double getPrice () {
        return price;
    }

    public int age (){
        return 2020-yearModel;
    }

    public String getColor ()
    {
        return color;}

    public  String toString () {
        return ("SittingCapacity: " + sittingCapacity + "\nManufacturer: "
+ manufacturer + "\nprice: " + price +
        "\nYear Model: " + yearModel + "\nColor: " + color);}
} //end of class Car
```

(C) Write a Java application called **CarExample** having only main method to test your class Car. Create an object myCar of type Car having following values of data fields:
sittingCapacity = 5; manufacturer = “BMW”; price = 15400; yearModel = 2016; color = “Red”;

```
public class CarExample {
public static void main(String[] args) {
        Car mycar = new Car(5,”BMW”,15400,2016,”Red”);
        System.out.println(mycar.toString());
        System.out.println(mycar.age());
    }
}
```

Output

```
SittingCapacity: 5
Manufacturer:BMW
price: 15400.0
Year Model: 2016
Color: Red
4
```

Questions from previous exams

Question (1):

Assume that **ActionListener** is an interface. We want to write a class called **ButtonHandler** that implements the interface ActionListener. Which of the following is the correct heading of class **ButtonHandler**:

(a) public class ButtonHandler extends ActionListener

(b) public class ButtonHandler implements ActionListener

(c) public class ButtonHandler interface ActionListener

(d) public class ButtonHandler abstract ActionListener

Question (2):

Assume that **Employee** is an interface and **RegularEmployee** is an actual class (concrete class) that implements the interface Employee.

Which of the following is **NOT** a valid statement:

(a) Employee e1 = new Employee(Ali, 15012);

(b) Employee e2 = new RegularEmployee(Ali, 15012, 1050);

(c) RegularEmployee e3 = new RegularEmployee(Ali, 15012, 1050);

(d) Employee e4;