Day 5

- In java, checking array bounds is a job of JVM.
- Using illegal index, if we try to access element from array then JVM throws ArrayIndexOutOfBoundsException.

```
int[] arr = new int[ ] { 10, 20, 30 };
//int element = arr[ -1  ]; //ArrayIndexOutOfBoundsException
int element = arr[ arr.length  ];//ArrayIndexOutOfBoundsException
```

- foreach loop is also called as iterator in java.
- It is used to traverse collection in forward direction only. During travsering, we can only read the values.

Array of value type

```
boolean[] arr = new boolean[ 3 ];
```

- If we create array of value type then default value of array depends on default value of data type.

Array of references

```
//Single dynamic object in C++
Complex *ptr = new Complex( );
//Array of objects in C++
Complex *ptr = new Complex[ 3 ];
```

```
Complex c1;
//c1 is object reference / reference
Complex c1 = new Complex();
//It is instantiation of single java instance

Complex[] arr;
//arr is reference of single dimensional array

Complex[] arr = new Complex[ 3 ];
//It is array of references in java whose default value is null.
```

Array of instances

```
Complex c1 = new Complex();
Complex c2 = new Complex();
```

```
Complex c3 = new Complex();
```

```
Complex[] arr = new Complex[ 3 ];
arr[ 0 ] = new Complex();
arr[ 1 ] = new Complex();
arr[ 2 ] = new Complex();
```

```
Complex[] arr = new Complex[ 3 ];
for( int i = 0; i < arr.length; ++ i )
    arr[ i ] = new Complex();
```

Passing argument by reference:

- In java, we can pass argument to the method by value only.
- Using array, we can pass, argument to the method by reference.

```
private static void swap(int[] arr)
{
    int temp = arr[ 0 ];
    arr[ 0 ] = arr[ 1 ];
    arr[ 1 ] = temp;
}
public static void main(String[] args)
{
    int a = 10;
    int b = 20;

    int[] arr = new int[ ] { a, b };
    Program.swap( arr );
    a = arr[ 0 ]; b = arr[ 1 ];

    System.out.println("a       :       "+a);
    System.out.println("b       :       "+b);
}
```

Variable argument method

```
private static void sum( int... args )
{
    int result = 0;
    for( int element : args )
        result = result + element;
    System.out.println("Result  :       "+result);
}
```

- public static String format(String format, Object... args);
- public PrintStream printf(String format, Object... args);
- public Object invoke(Object obj, Object... args);

System Date and Time

- Using Calendar:
  - Calendar is abstract class declared in java.util package.
  - Fields:
    1. public static final int DATE
    2. public static final int MONTH
    3. public static final int YEAR
    4. public static final int HOUR
    5. public static final int MINUTE
    6. public static final int SECOND
  - Methods
    1. public static Calendar getInstance()
    2. public int get(int field)

```
Calendar c = Calendar.getInstance();
int day = c.get(Calendar.DATE);
int month = c.get(Calendar.MONTH) + 1;
int year = c.get(Calendar.YEAR);
```

- Using Date:
  - It is a concrete class declared in java.util package.
  - It is Deprecated class.
  - If we want to format Date and Time then we should use SimpleDateFormat class which is declared in java.text package.

```
//Date date = new Date(119, 10, 5);
Date date = new Date();
String pattern = "dd/MM/yyyy";
SimpleDateFormat sdf = new SimpleDateFormat(pattern);
String strDate = sdf.format(date);
```

- Using LocalDate:
  - It is a final class declared in java.time package.

```
LocalDate ld = LocalDate.now();
int day = ld.getDayOfMonth();
int month = ld.getMonthValue();
int year = ld.getYear();
```

Hierarchy

- It is major pillar of oops.
- level / order / ranking of abstraction is called hierarchy.
- Purpose of hierarchy is to achieve reusability.
- Type of hierarchy:
    1. has-a : Association
    2. is-a : Inheritance
    3. use-a : Dependancy
    4. creates-a: Instantiation

Association

- If "has-a" relationship exist between two types then we should use association.
- Example:
    1. Car has-a engine
    2. Room has-a wall
- If object/instance is part of/ component of another instance then it is called association.
- Composition and aggregation are specialized form of association.

```
class Date
{   };
class Address
{   };
class Person
{
    string name;    //Association
    Date dob;       //Association
    Address currAddress; //Association
};
Person p;
```

```
class Date
{   };
class Address
{   };
class Person
{
    string *name;    //Association
    Date *dob;       //Association
    Address *currAddress; //Association
public:
    Person( )
    {
        this->name = new string();
        this->dob = new Date();
```

```
        this->currAddress = new Address();
    }
};
Person *ptr = new Person();
```

```
class Date
{
    private int day;
    private int month;
    private int year;
}
class Address
{
    private String city;
    private String state;
    private int pincode;
}
class Person
{
    private String name;
    private Date dob;
    private Address currAddress;
    public Person( )
    {
        this.name = new String();
        this.dob = new Date();
        this.currAddress = new Address();
    }
}
Person p = new Person();
```

- In java, object/instance do not contain another instance directly. In other words, association do not represent physical containment.

Modularity

- It is major pillar of oops
- It is used to minimize module dependancy.
- In java we can achive it using .jar, .war, .ear file.
- jar file is reusable component of java i.e if we want to create reusable library then we should create jar file.
- if we want to create jar file then we should use jar tool.
- AssociationLib(Project) : .jar
    - org.sunbeam.dac.lib(Package)
        1. Date
        2. Address
        3. Person
- AssociationTest(Project)
    - org.sunbeam.dac.test(Package)

    1. Program
- Steps to create jar file
    1. Create java project (AssociationLib) and define types inside it.
    2. Right click on java project -> Export -> java -> jar file -> Next -> select type -> choose location for jar file -> click on finish.
- Steps to use jar file / steps to add jar file in classpath / runtime classpath / buildpath.

Inheritance

- If "is-a" relationship exist between two types then we should use inheritance.
- Example:
    1. Car is vehicle
    2. Water is liquid
- Without modifying implemetation of existing class if we want to extend meaning of that class then we should use inheritance.
- In java parent class is called super class and child class is called sub class.
- If we want extend class or create sub class then we should use extends keyword.

```
class Person
{   }
class Employee extends Person
{   }
```

- Java do not support private and protected mode of inheritance. Hence default mode of inheritance is public.
- In java, class can extend only one class.
- Except constructor, all the members of super class (including private and static) inherit into sub class.
- all the fields of super class inherit into sub class but only non static field get space inside instance.
- If functionality of super class method is logically incomplete then we should redefine that method in sub class.
- Inside method of sub class, if we want to access members of super class then we should use super keyword.
- From any constructor of sub class, by default, super class's parameterless constructor gets called. If we want to call any constructor of super class from constructor of sub class then we should use super statement.
- super statement must be first statement inside constructor body.
- During inheritance, members of super class inherit into sub class. Hence sub class instance can be considered as super class instance.
- Since sub class instance can be considered as super class instance, we can use it in place of super class instance.

```
Person p = new Person();    //Ok
Person p = new Employee();  //Ok : Upcasting
```

- Members of sub class do not inherit into super class. Hence super class instance can not be considered as sub instance.
- Since super class instance can not be considered as sub class instance, we can not use it in place of sub class instance.

```
Employee emp = new Employee();  //Ok
Employee emp = new Person( );   //Not OK
```

Typing:

- It is minor pillar of oops.
- It is also called as polymorphism
- Ability of an object/instance to take multiple forms is called polymorphism
- In java we can achive polymorphism using
    1. Method overloading
    2. Method overriding
- Using polymorphism, we can reduce maintenance of system.

Upcasting

- We can convert reference of sub class into reference of super class. It is called upcasting.
- Upcasting is used to minimize object dependancy in the code.

```
Employee emp = new Employee("ABC",23,1672,35000);
//Person p = ( Person)emp;      //Upcasting
Person p = emp; //Upcasting
```

- In above code, emp has access to all the members of super class as well as sub class. But p has access to only members of super class.

Downcasting

- We can not convert, reference of super class into reference of sub class. It is called downcasting.
- In case of downcasting, explicit typecasting is mandatory.

```
Person p=new Employee("ABC",23,1672,35000);
p.showRecord();
Employee emp = (Employee) p;//Downcasting
emp.displayRecord();
```

```
Person p = null;
System.out.println(p);//null
Employee emp = (Employee) p;//Downcasting
System.out.println(emp);//null
```

- If downcasting fails then JVM throws ClassCastException.

```
Person p = new Person();
Employee emp = (Employee) p;//Downcasting
//Output : ClassCastException
```

Dynamic Method dispatch

- In java, all the methods are by default virtual.
- In case of upcasting, process of calling method of sub class using reference of super class is called dynamic method dispatch(in C++ : runtime polymorphism)

```
class Person
{
        public void printRecord( )
        {   }
}
class Employee extends Person
{
        public void printRecord( )
        {}
}
public class Program
{
        public static void main(String[] args)
        {
                Person p = new Employee();
                p.printRecord(); //DMD
        }
}
```

Method overriding

- Process of redefining method of super class inside sub class is called method overriding.
- Rules of method overriding

1. Access modifier in sub class method should be same or it should be wider.
2. Return type in sub class method should be same or it should be sub type.
3. Method name, number of parameters and type of parameters in sub class method must be same.
4. Checked exception list in sub class method should be same or it should be sub set.

Final method

- If implementation method is logically 100% complete then we should declare such method final.
- Final method inherit into sub class but we can not override it in sub class.
- In java, we can not override following methods:

1. constructor
2. private method
3. static method
4. final method

- Overridden method can be declared as final.
- Examples:
    1. getClass
    2. wait
    3. notify
    4. notifyAll

Abstract method

- If implementation method is logically 100% incomplete then we should declare such method abstract.
- abstract is keyword in java.
- abstract method do not contain body.
- if we delare method abstract then it is mandatory to declare class abstract.
- Without declaring method abstract, we can declare class abstract.
- We can not instantiate abstract class. But we can create reference of it.
- Example:

    1. java.lang.Number
    2. java.lang.Enum
    3. java.util.Calendar
    4. java.util.Dictionary
    5. java.io.InputStream
    6. java.io.OutputStream

- It is mandatory to override abstract method in sub class otherwise sub class can be considered as abstract.
- If we extend abstract class then either we should override method in sub class or we should declare sub class abstract.

Final class

- If implementation of the class is logically 100% complete then we should declare such class final.
- We can not extend final class.
- Example:

1. java.lang.System
2. java.lang.Math
3. java.lang.String, StringBuffer, StringBuilder
4. All Wrapper classes
5. java.util.Scanner

Sole Constructor

- Constructor of super class, that is designed to call from constructor of sub class only is called sole constructor.

```
abstract class A
{
        private int num1;
        private int num2;
        public A( int num1, int num2 ) //Sole Constructor
        {
                this.num1 = num1;
                this.num2 = num2;
        }
        public void printRecord( )
        {
                System.out.println("Num1        :       "+this.num1);
                System.out.println("Num2        :       "+this.num2);
        }
}
class B extends A
{
        private int num3;
        public B( int num1, int num2, int num3 )
        {
                super( num1, num2 );
                this.num3 = num3;
        }
        public void printRecord( )
        {
                super.printRecord();
                System.out.println("Num3        :       "+this.num3);
        }
}
public class Program
{
        public static void main(String[] args)
        {
                B b = new B(10,20, 30 );
                b.printRecord();
        }
}
```

instanceof

- It is operator.
- At runtime, if we want to check inheritance then we should use instanceof operator.
- It returns boolean value.

```
private static void accept(Shape shape)
{
    if( shape instanceof Rectangle )
    {
        Rectangle rect = (Rectangle) shape;
    }
    else
    {
        Circle c = (Circle) shape;
    }
}
```