Day 5

Hierarchy

- It is major pillar oops
- Order / ranking of abstraction is called hierarchy.
- Hierachy is used to achieve reusability
- Types of hierarchies
    1. has-a / part-of : Association
    2. is-a / kind-of : Inheritance
    3. use-a : Dependancy
    4. creates-a : Instantiation

Association

- Example
    1. Car-has a engine
    2. Room has-a chair
- If "has-a" realtionship exist between two types then we should use Association.
- Example
    - Person has-a name
    - Person has-a birthdate
    - Person has-a address
- If object/instance is part of or component of another instance then it is called association.

Inheritance

- Example:
    1. Employee is-a person
    2. Book is-a product
    3. Rectangle is-a Shape
- If is-a relationship exist between two types then we should use inheritance.

```
//Parent class / Super class
class Person
{   }
//Child class / Sub class
class Employee extends Person
{   }
```

- Except constructor, all the members(fields[static + non static], methods[static + non static], Nested type[Inteface, class, enum]) of super class inherit into sub class.
- Members of sub class do not inherit into super class but members of super class inherit into sub class.
- During inheritance, all the fields of super class inherit into sub class but only non static field get space inside instance of sub class.

- Note : Private fields inherit into sub class.
- All the methods of super class inherit into sub class.
- Process of acquiring properties and behavior of super class inside sub class is called inheritance.
- To create sub class we should use extends keyword.
- If implementation of super class method is logically incomplete then we should redefine method in sub class.
- If we want to access members of super class inside method of sub class then we should use super keyword.
- If we create instance of sub class then first super class and then sub class constructor gets called.
- By default, from sub class contructor, super class's parameterless constructor gets called. If we want to call any contructor of super class from constructor of sub class then we should use super statement.
- Super statement must be first statement in constructor body.
- 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 p1 = null;    //OK
Person p2 = new Person();   //OK
Person p3 = new Employee( );    //OK : Upcasting
```

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

```
Employee e1 = null; //OK
Employee e2 = new Employee(); //OK
Employee e3 = new Person( ) ; //Not OK
```

- Super class reference can contain reference of sub class instance. This process of converting referece of sub class into super class is called upcasting.
- If we want to minimize object dependancy in the code then we should use upcasting.

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

- or

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

- Process of converting reference of super class into reference of sub class is called downcasting.

```
Person p = null;
Employee emp = ( Employee)p;    //OK : Downcasting
s.o.p( p )//null
s.o.p( emp )//null
```

```
Person p = new Employee(); //Upcasting
Employee emp = ( Employee)p;    //OK : Downcasting
```

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

- If downcasting fails then JVM throws ClassCastException

```
Number n1 = new Integer( 125 );//Upcasting
Integer n2 = ( Integer )n1; //OK : Downcasting
```

```
Number n1 = new Double( 3.14 ); //Upcasting
Double n2 = ( Double ) n1; //OK : Downcasting
```

```
Number n1 = new Double( 3.14 ); //Upcasting
Integer n2 = ( Integer ) n1; //Downcasting : ClassCastException
```

- In java, in case upcasting, using super class reference, we can access method of sub class i.e methods in java are by default virtual.

```
Person p = new Employee( )//Upcasting
p.printRecord( ); //Dynamic Method Dispatch
```

- Process of calling method sub class using reference of super class is called dynamic method dispatch.
- Process of redefining method of super class inside sub class is called method overriding.
- If we want to compare state of instance of value type then we should use operator ==.

```
public static void main(String[] args)
{
    int num1 = 10;
    int num2 = 10;
    if( num1 == num2 )
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Equal
}
```

- If we want to compare state of references then we should use operator ==

```
public static void main(String[] args)
{
    Employee emp1 = new Employee("Sandeep", 33, 45000);
    Employee emp2 = new Employee("Sandeep", 33, 45000);
    if( emp1 == emp2 )
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Not Equal
}
```

- If we want to compare state of instances then we should use equals method.
- "equals" is a non final method of java.lang.Object class.
- Syntax:

> public boolean equals(Object obj);

- If we do not define equals method inside class then its's super class's equals method gets called. If any class do not contain equals method then Object class's equals method gets called.

```
public boolean equals(Object obj)
{
    return (this == obj);
}
```

- equals method of Object class do not compare state of instances rather it compares state of references. If we want to compare state of instances then we should override equals method inside sub class.

Exception Handling

- AutoCloseable is interface declared in java.lang package. "void close()throws Exception" is a method of AutoCloseable interface.
- Closeable is sub interface of AutoCloseable interface which is declared in java.io package.
- "void close() throws IOException" is a method of Closeable interface.

```
class File implements AutoCloseable
{
        @Override
        public void close() throws Exception
        {       }
}
public class Program
{
        public static void main(String[] args)
        {
                File file = new File();
        }
}
```

- In context of exception handling, if class implements java.lang.AutoCloseable/java.io.Closeable interface the such class is called Resource clas/Type and its instance is called resource.

```
public char charAt(int index)
{
    if ((index < 0) || (index >= value.length))              throw new
StringIndexOutOfBoundsException(index);
    return value[index];
}
```

```
String str = new String("SunBeam");
char ch = str.charAt( str.length() ); //StringIndexOutOfBoundsException
```

- Exception is an object/instance which is used to send notification to the end user of the system if any exceptional situation occurs in the program.
- We should handle exception
    1. To handle all runtime errors centrally.
    2. To handle OS resources carefully.
- Following are operating system resources / non java resources
    1. File
    2. Thread
    3. Socket
    4. Network connection
    5. IO devices
- If we want to handle exception then we should use five keywords
    1. try
    2. catch
    3. throw
    4. throws
    5. finally

- Throwable is a class declared in java.lang package. It is super class of Error and Exception.
- Consider code in C++

```
int calculate( int num1, int num2 )
{
    if( num2 == 0 )
    {
        //throw 0;    //OK
        //throw 10.5;    //OK
        //throw "Exception";    //OK
        ArithmeticException ex("/ by zero exception");
        throw ex;    //OK
    }
    return num1 / num2;
}
```

- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the JVM or can be thrown by the Java throw statement.
- Similarly, only Throwable class or one of its subclasses can be the argument type in a catch clause.
- Throwable constructors

1. public Throwable()

```
Throwable th1 = new Throwable( );
```

2. public Throwable(String message)

```
Throwable th1 = new Throwable( "Exception message" );
```

3. public Throwable(Throwable cause)

```
Throwable cause = new Throwable( );
Throwable th2 = new Throwable( cause );
```

4. public Throwable(String message, Throwable cause)

```
Throwable cause = new Throwable( );
Throwable th2 = new Throwable( "Ex Message", cause );
```

- Methods of Throwable

1. public String getMessage()
2. public Throwable getCause()
3. public void printStackTrace()

Error

- It is a sub class of java.lang.Throwable class
- Error gets generated due to environmental condition or due to failure of Runtime environment.
- Example
    1. Virtual machine internal error
    2. Hard disk crash
    3. Out Of memory problem
- We can not recover from error.
- We can use try-catch block to handle error but we should not handle it.
- Example
    1. VirtualMachineError
    2. InternalError
    3. StackOverflowError
    4. OutOfMemoryError

Exception

- It is a sub class of java.lang.Throwable class.
- Exception gets generated due to application.
- We can recover from exception.
- We should write try catch block to handle exception
- e.g.
    1. ClassNotFoundException
    2. IOException
    3. SQLException
    4. NumberFormatException

Types of exception

1. Checked Exception
2. Unchecked Exception

- above types of exceptions are designed for java compiler.

Unchecked Exception

- java.lang.RuntimeException and all its sub classes are considered as uncheked exception.
- It is not mandatory to handle unchecked exception.
- Example
    1. NullPointerException
    2. NumberFormatException
    3. NegativeArraySizeException
    4. ArrayIndexOutOfBoundsException

5. StringIndexOutOfBoundsException
6. ArrayStoreException
7. ClassCastException
8. IllegalArgumentException

Checked Exception

- java.lang.Exception and all its sub classes exception RuntimeException and all its sub classes are called checked exception classes.
- It is mandatory to handle checked exception.
- Example
    1. ClassNotFoundException
    2. IOException
    3. SQLException
    4. CloneNotSupportedException
- While performing, arithmetic operation, if any exceptional situation occurs then JVM throws ArithmeticException. For example, an integer "divide by zero".
- If we want to inspect group of statements for exception then we should use try block/handler.
- We can write try block after catch/ finally block.
- Try block must have at least one catch/finally block or resource.
- If we want to handle exception then we should use catch block/handler.
- Catch block can handle exception thrown from try block.
- We can not define catch block before try and after finally block.
- Single try block may have multiple catch block.
- In java, in single catch block, we can handle multiple specific exceptions. Such catch block is called multi catch block.

```
try
{   }
catch (ArithmeticException | InputMismatchException ex)
{
    ex.printStackTrace();
}
```

- NullPointerException is unchecked exception

```
NullPointerException ex = new NullPointerException();//OK
RuntimeException ex = new NullPointerException();//OK
Exception ex = new NullPointerException();//OK
```

- InterruptedException is checked exception

```
InterruptedException ex = new InterruptedException();//OK
Exception ex = new InterruptedException();//OK
```

- Exception class reference variable can contain reference of any checked as well as uncheked exception. Hence to write generic catch block we should use java.lang.Exception class.

```
try
{   }
catch(Exception ex )
{
    ex.printStackTrace();
}
```

- If child/parent relation is exist between exception type then we must handle child type exception first.
- If we want to generate new exception then we should use throw keyword.
- throw statement is a jump statement.
- finally is a block, which is used to release local resources.
- We can not write finally block before try block and catch block.
- For single try block we can write only one finally block.
- JVM always execute finally block.

try with resource

- resource type must impletement AutoCloseable /Closeable interface.

```
try( Scanner sc = new Scanner(System.in);)
{   }
catch (Exception ex )
{
    ex.printStackTrace();
}
```

- throws is a keyword.
- If we want to delegate exception from one method to another then we should use throws clause.

```
public static void print( ) throws InterruptedException
{
    for( int count = 1; count <= 10; ++ count  )
    {
        System.out.println("Count       :       "+count);
        Thread.sleep(500 );
    }
}
```

- A, B, C : Checked Exceptions

```
//public static void print( int number )throws A, B, C
public static void print( int number )throws Exception
{
    if( number <  0 )
        throw new A();
    else if( number >= 0 && number < 10 )
        throw new B();
    else if( number >= 10 && number < 20 )
        throw new C();
    else
        System.out.println( number );
}
```

Exception Chaining

- Process of handling exception by throwing new type of exception is called exception chaining.

```
class A
{
        public void print( )
        {       }
}
class B extends A
{
        @Override
        public void print() throws RuntimeException
        {
                try
                {
                        for( int count = 1; count <= 10; ++ count )
                        {
                                System.out.println("Count       :
"+count);
                                Thread.sleep(100);
                        }
                }
                catch (InterruptedException cause)
                {
                        throw new RuntimeException( cause );  //Exception
chaining
                }
        }
}
```