Day 12

- In C++, in case of upcasting, if we want to call function depending on type of object rather than type of pointer/reference then we should declare function in base class virtual.
- In java, all the methods, are by default virtual.
- Process of calling method of sub class using referene of super class is called dynamic method dispatch.
- Method overriding: Process of redefining method of super class inside sub class is called method overriding.
- Rules of method overriding:

1. Access modifier of 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(class/interface).
3. Name of the method, 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.

- Override is annotation which help developer to override method.
- We can not override following methods in java:
    1. Constructor
    2. static method
    3. final method
    4. private method

```
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/value or variable of primitive type then we should use == operator.

```
Employee emp1 = new Employee("Abc", 10, 15000);
Employee emp2 = new Employee("Abc", 10, 15000);
if( emp1 == emp2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output    :   Not Equal
```

- If we want to compare state of references then we should use == operator.
- If we want to compare state of instances then we should use equals method.
- equals is method of java.lang.Object class.
- Syntax: public boolean equals(Object obj);

- Implementation of equals from Object class

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

```
public static void main(String[] args) {
    Employee emp1 = new Employee("Abc", 10, 15000);
    Employee emp2 = new Employee("Abc", 10, 15000);
    if( emp1.equals(emp2))  //By default Object Class's equals will call
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output    :   Not Equal
    }
```

- If we do not define equals method in a class then super class's equals method gets called.
- equals method of java.lang.Object class do not comapare state of instances, rather it compares state of references. If we want to compare state of instances then we should override method in a class.

```
@Override
    public boolean equals(Object obj) {
        if( obj != null )
        {
            Employee other = (Employee) obj;
            if( this.empid == other.empid )
                return true;
        }
        return false;
    }
```

```
Employee emp1 = new Employee("Abc", 10, 15000);
Employee emp2 = new Employee("Abc", 10, 15000);
if( emp1.equals(emp2))
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output    : Equal
```

Final Method

- If implementation of a method is logically 100% complete then we should declare that method final.
- We can not override final method in sub class but it inherit into sub class.
- Example:

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

Abstract Method

- abstract is a keyword in java.
- If implementation of a method is logically 100% incomplete then we should declare that method abstract.
- Abstract method do not contain body.
- If method is abstract then it is mandatory to declare class abstract.
- Without declaring method abstract, we can declare class abstract.
- Method 1

```
abstract class A{
    public abstract void f1();
}
class B extends A{
    @Override
    public void f1( ){  //OK
        //TODO
    }
}
```

- Method 2

```
abstract class A{
    public abstract void f1();
}
abstract class B extends A{
}
```

- It is mandatory to overide abstract abstract method in sub class otherwise sub class will be considered as abstract.
- Example
    - Methods of java.util.Dictionary class are abstract.

Abstract Class

- We can not instantiate abstract class but we can create reference of abstract class.
- It can contain:
    1. Nested type
    2. Field[static as well as non static]
    3. Constructor
    4. Concrete method

5. Asbtract method

- Example:
  - java.lang.Number
  - java.lang.Enum
  - java.util.Calendar
  - java.util.Dictionary

Final Class

- If implementation of a class is logically 100% complete then we should declare such class final.
- We can not extend final class i.e we can not create sub class of final class.
- Final class is a concrete class hence we can instantiate it.
- Example:
  - java.lang.System
  - All wrapper classes
  - java.lang.Math
  - java.lang.String, StringBuffer, StringBuilder
  - java.util.Scanner

Exception Handling

- Following are OS resources that we use in the program
  1. Memory
  2. File
  3. Thread
  4. Scoket
  5. Network Connection
  6. IO devices etc
- OS resources are limited hence we should use it carefully. To avoid their leakage we should use exception handling
- Exception is an instance which is used to send notification to the end user of the system if exceptional situation occurs in the program.
- If we want to handle exception then we should use exception handling mechanism
- To handle exception, we should use 5 keywords
  1. try
  2. catch
  3. throw
  4. throws
  5. finally
- Throwable is a class declared in java.lang package.
- 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.
- The Throwable class is the superclass of all errors and exceptions in the Java language.