Agenda

```
Java 7 Interface
Types of interface inheritance
Marker Interface
Date/LocalDate/Calender class
Exception Handling
```

Interface (Java 7)

```
- It is a set of protocol  that is used to keep the method designs same
across all the related and non related classes
- To declare an interface we have to use interface keyword.
- Interface is a blueprint of a class
- All classes can implement the interface they cannot extend the interface
- We cannot create object of the interface but we can create reference of
the interface.
- By default all the methods inside interface are public
- If we declare the field inside interface then they are by default public
static final.
```

Difference between Abstract & Interface

Abstract class

```
- We can have abstract as well as non abstract methods in abstract class
- We can write constructor inside abstract class
- A class can extends the abstract class
- When the classes are of related types then use abstract class
- When state is involved use abstract class
```

Interface

```
- All the methods are by default abstract
- we cannot write constructor inside interface
- A class can implements the interface
- When classes are of non related types then use interface
- when state is not involved use interface
```

Types of interface Inheritance

```
1. Single Inheritance
2. Hirerachical Inheritance
3. Multilevel Inheritance
4. Multiple Inheritance
```

Marker Interface

```
- An empty interface is called as Marker Interface
```

Date

```
- Date is class which will provide the system date and time.
- Date can be created with the default Ctor. Other ctor's are deprecated
- It is recommended to use calender class
```

```
        Date d1 = new Date();
        System.out.println(d1);

        //Deprecated
        //Date d2 = new Date(1,1,2000);
        //System.out.println(d2);
```

Calender

```
- Calender class is a mutable class
- to create the object of calender class you have to use the getInstance
method of calender class.
- Calender calender = Calender.getInstance();
- To fetch the individual day,month,year,etc... we have to use a get(int)
which takes an int value representing the filed.
- sysout(calender.get(Calender.DATE))
```

```
Calendar calendar= Calendar.getInstance();
System.out.println(calendar);
System.out.println(calendar.get(Calendar.DATE));
System.out.println(calendar.get(Calendar.MONTH));
System.out.println(calendar.get(Calendar.YEAR));

System.out.println("Date = "+calendar.get(Calendar.DATE)+"/"+
(calendar.get(Calendar.MONTH)+1+"/"+calendar.get(Calendar.YEAR)));
```

```
calendar.set(Calendar.MONTH, 5);

System.out.println("Date = "+calendar.get(Calendar.DATE)+"/"+
(calendar.get(Calendar.MONTH)+1+"/"+calendar.get(Calendar.YEAR)));
```

LocalDate

```
- LocalDate class is a immutable class
```

```
LocalDate localDate = LocalDate.now();
System.out.println(localDate);
LocalDate localDate2 = LocalDate.of(2000, 1, 1);
System.out.println(localDate2);

LocalDate localDate3 = LocalDate.of(calendar.get(Calendar.YEAR),
calendar.get(Calendar.MONTH)+1, calendar.get(Calendar.DATE));

System.out.println(localDate3);
```

Exception Handling

```
- To Seperate the business logic from error/exception handling logic we use
this mechanism.
- Throwable is the Super class of all Errors and Exception

1. Error
    - Errors are occured due to Runtime Environment
    - You should not handle the errors
    - We can try to handle the error but it is not recommended as we cannot
recover from those errors

2. Exception
    - Exceptions are occured due to the appication
    - You should always try to handle those exceptions
    - Exception class and all its subclasses except RunTimeException class
all are considered as checked Exception
    - RuntimeException and all its subclasses are called as unChecked
Exception
    - Checked Exceptions are mandatory to handle
    - Unchecked Exceptions are optional to handle
```

Exception Handling Mechanism

```
1. try
2. catch
3. throw
4. throws
5. finally
```

lab flow

```
Interface
Complete the assignments
Exception Handling Theory.
```