Agenda

- Exception Handling
  - Exceptions
  - Errors
  - Exception Chaning
  - Custom Exceptions
- ~~Date/LocalDate/Calender~~

Exception Handling

- Exceptions represents runtime problems
- If not handled in the current method it is sent back to the calling method.
- To perform exception handling java have provided below keywords
  - try
  - catch
  - throw
  - throws
  - finally
- Java operators/APIs throw pre-defined exception if runtime problem occurs.
- Exceptions need to handled otherwise the it terminates the program by throwing that exception.
- we use try catch block to handle the exception. Inside try block keep all the method calls that generate exception and handle the exception inside catch block
- When exception is raised, it will be caught by nearest matching catch block. If no matching catch block is found, the exception will be caught by JVM and it will abort the program.
- We can handle multiple exceptions in a single catch block.
- when exceptions are generated if we dont to handle it program terminates,however the resources that are in used should be closed.
- the resources can be closed in finally block that can be used with a try block.
- if the classes have implemented AutoCloseable interface we can also use try with resource with the try block.
- when we use try block then,it should atleast have
  - 1. a catch block
  - 2. a finally block
  - 3. try with resource
- Java have divided the exceptions into two categories
  - 1. Error

    - 2. Exception

- java.lang.Throwable is the super class of all the Errors and Exceptions

```
- Throwable (Super class of all Errors and Exceptions)
    - Error
        - VirtualMachineError
            - OutOfMemoryError
            - StackOverflowError
        - IOError
    - Exception (Checked Exception - Checked at compile time,forced to handle)
        - CloneNotSupportedException
        - IOException
        - InterruptedException
        - RuntimeException (Unchecked Exception - Not Checked By Compiler)
            - ArithmeticException
            - ClassCastException
            - IndexOutOfBoundsException
                - ArrayIndexOutOfBoundsException
                - StringIndexOutOfBoundsException
            - NegativeArraySizeException
            - NoSuchElementException
                - InputMismatchException
            - NullPointerException
```

Errors

- Errors are generated due to runtime environment.
- It can be due to problems in RAM/JVM for memory management or like crashing of harddisk, etc.
- We cannot recover from such errors in our program and hence such errors should not be handled.
- we can write a try catch block to handle such errors but it is recommended not to handle such errors.

Exceptions

- Exception class and all its sub classes except Runtime exception class are all Checked Exception
- Runtime Exception and all its sub classes all unchecked exceptions
- Checked exceptions are mandatory to handle.

Exception Handling Keywords

- 1. try
    - Code where runtime problems may arise should be written in try block.
- 2. catch
    - Code to handle error/exception should be written in catch block.
    - Argument of catch block must be Throwable or its sub-class.
    - Generic catch block -- Performs upcasting -- Should be last (if multiple catch blocks)
- 3. finally
    - Resources are closed in finally block.
    - Executed irrespective of exception occurred or not.

    - finally block not executed if JVM exits (System.exit()).
- 4. "throw" statement
    - Throws an exception/error i.e. any object that is inherited from Throwable class.
    - Can throw only one exception at time.
    - All next statements are skipped and control jumps to matching catch block.
- 5. throws
    - Written after method declaration to specify list of exception not handled by called method and to be handled by calling method.
    - Writing unhandled checked exceptions in throws clause is compulsory.
    - Sub-class overridden method can throw same or subset of exception from super-class method.

Exception chaining

- Sometimes an exception is generated due to another exception.
- For example, database SQLException may be caused due to network problem SocketException.
- To represent this an exception can be chained/nested into another exception.
- If method's throws clause doesn't allow throwing exception of certain type, it can be nested into another (allowed) type and thrown.

```
public static void getEmployees() throws SQLException {
        // logic to get all the employees from database
        boolean connection = false;
        if (connection) {
            // fetch data
            boolean data = false;
            if (data)
                System.out.println(data);
            else
                throw new SQLException("Data not found");
        } else
            throw new SQLException("failed", new SocketException("Connection
rejected"));
    }
```

User defined exception class

- If pre-defined exception class are not suitable to represent application specific problem, then user-defined exception class should be created.
- User defined exception class may contain fields to store additional information about problem and methods to operate on them.
- Typically exception class's constructor call super class constructor to set fields like message and cause.
- If class is inherited from RuntimeException, it is used as unchecked exception. If it is inherited from Exception, it is used as checked exception.