Agenda

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

Interface (Java 7 or Earlier)

- Interfaces are used to define standards/specifications.
- A standard/specification is set of rules.
- Interfaces are immutable i.e. once published interface should not be modified.
- Interfaces contains only method declarations. All methods in an interface are by default abstract and public.
- They define a "contract" that is must be followed/implemented by each sub-class.
- Interfaces enables loose coupling between the classes i.e. a class need not to be tied up with another class implementation.
- Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces.
- Java 7 interface can only contain public abstract methods and static final fields (constants).
- They cannot have non-static fields, static methods, and constructors.
- Examples:
    - java.io.Closeable / java.io.AutoCloseable
    - java.lang.Runnable
- Multiple interface inheritance is allowed in Java

```
interface Displayable {
void display();
}
interface Acceptable {
void accept();
}
class Employee implements Displayable,Acceptable{}
```

- If two interfaces have same method, then it is implemented only once in sub-class.

class vs abstract class vs interface

- class
    - Has fields, constructors, and methods
    - Can be used standalone -- create objects and invoke methods

    - Reused in sub-classes -- inheritance
    - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism

- abstract class

    - Has fields, constructors, and methods
    - Cannot be used independently -- can't create object
    - Reused in sub-classes -- inheritance -- Inherited into sub-class and must override abstract methods
    - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism

- interface

    - Has only method declarations
    - Cannot be used independently -- can't create object
    - Doesn't contain anything for reusing (except static final fields)
    - Used as contract/specification -- Inherited into sub-class and must override all methods
    - Can invoke overridden methods in sub-class using super-class reference -- runtime polymorphism

Marker interfaces

- Interface that doesn't contain any method declaration is called as "Marker interface".
- These interfaces are used to mark or tag certain functionalities/features in implemented class.
- In other words, they associate some information (metadata) with the class.
- Marker interfaces are used to check if a feature is enabled/allowed for the class.
- Java has a few pre-defined marker interfaces. e.g. Serializable, Cloneable, etc.
    - java.io.Serializable -- Allows JVM to convert object state into sequence of bytes.
    - java.lang.Cloneable -- Allows JVM to create copy of the class object.

Cloneable interface

- Enable creating copy/clone of the object.
- If a class is Cloneable, Object.clone() method creates a shallow copy of the object.
- If class is not Cloneable, Object.clone() throws CloneNotSupportedException.
- A class should implement Cloneable and override clone() to create a deep/shallow copy of the object.

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.