Agenda

- Exception Handling
  - Exceptions
  - Errors
  - Custom Exceptions
- clone()
- Enum
- ~~Garbage Collector~~
- ~~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

- There are two types of exceptions

1. Checked exception -- Checked by compiler and forced to handle
2. Unchecked exception -- Not checked by compiler

- 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

Exception Handling Keywords

- 1. try
    - Code where runtime problems may arise should be written in try block.
    - try block must have one of the following

        - catch block
        - finally block
        - try-with-resource
    - it Can be nested in try, catch, or finally 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.

1. Checked exception

- Exception and all its sub classes (except RuntimeException) are checked exceptions.
- Typically represents problems arised out of JVM/Java i.e. at OS/System level.
- IOException, SQLException, etc..
- Compiler checks if the exception is handled in one of following ways.
    - Matching catch block to handle the exception.

```
  void someMethod() {
try {
// file io code
    }
catch(IOException ex) {
// ...
    }
}
```

- throws clause indicating exception to be handled by calling method.

```
  void someMethod() throws IOException {
// file io code
}

void callingMethod() {
try {
```

```
            someMethod();
        }
    catch(IOException ex) {
    // ...
        }
    }
```

2. Unchecked exception

- RuntimeException and all its sub classes are unchecked exceptions.
- Typically represents programmer's or user's mistake.
- NullPointerException, NumberFormatException, ClassCastException, etc.
- Compiler doesn't provide any checks -- if exception is handled or not.
- Programmer may or may not handle (catch block) the exception. If exception is not handled, it will be caught by JVM and abort the application.

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.

Clone method

- The clone() method is used to create a copy of an object in Java. - It's defined in the java.lang.Object class and is inherited by all classes in Java.
- It returns a shallow copy of the object on which it's called.

```
protected Object clone() throws CloneNotSupportedException
```

- This means that it creates a new object with the same field values as the original object, but the fields themselves are not cloned.
- If the fields are reference types, the new object will refer to the same objects as the original object.
- In order to use the clone() method, the class of the object being cloned must implement the Cloneable interface.
- This interface acts as a marker interface, indicating to the JVM that the class supports cloning.
- It's recommended to override the clone() method in the class being cloned to provide proper cloning behavior.
- The overridden method should call super.clone() to create the initial shallow copy, and then perform any necessary deep copying if required.
- The clone() method throws a CloneNotSupportedException if the class being cloned does not implement Cloneable, or if it's overridden to throw the exception explicitly.

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.

Enum

- In C enums were internally integers
- In java, It is a keyword added in java 5 and enums are object in java.
- used to make constants for code readability
- mostly used for switch cases
- In java, enums cannot be declared locally (within a method).
- The declared enum is converted into enum class.
- The enum type declared is implicitly inherited from java.lang.Enum class. So it cannot be extended from another class, but enum may implement interfaces.
- The enum constants declared in enum are public static final fields of generated class.
- Enum objects cannot be created explicitly (as generated constructor is private).
- The enums constants can be used in switch-case and can also be compared using == operator.
- The enum may have fields and methods.

```
public abstract class Enum<E> implements java.lang.Comparable<E>,
java.io.Serializable {
    private final String name;
    private final int ordinal;

    protected Enum(String,int); // sole constructor - can be called from user-
defined enum class only

    public final String name(); // name of enum const
    public final int ordinal(); // position of enum const (0-based)
    public String toString(); // returns name of const
    public final int compareTo(E);// compares with another enum of same type on
basis of ordinal number
    public static <T> T valueOf(Class<T>, String);
    // ...
}
```

```
// user-defined enum
enum ArithmeticOperations {
    ADDITION, SUBTRACTION, MULIPLICATION, DIVISION
}

// generated enum code
final class ArithmeticOperations extends Enum {

private ArithmeticOperations(String name, int ordinal) {
    super(name, ordinal); // invoke sole constructor Enum(String,int);
}

public static ArithmeticOperations[] values() {
    return (ArithmeticOperations[])$VALUES.clone();
}

public static ArithmeticOperations valueOf(String s) {
    return (ArithmeticOperations)Enum.valueOf(ArithmeticOperations,s);
}


  public static final ArithmeticOperations ADDITION;
  public static final ArithmeticOperations SUBTRACTION;
  public static final ArithmeticOperations MULIPLICATION;
  public static final ArithmeticOperations DIVISION;
  private static final ArithmeticOperations $VALUES[];

  static {
    ADDITION = new ArithmeticOperations("ADDITION", 0);
    SUBTRACTION = new ArithmeticOperations("SUBTRACTION", 1);
    MULIPLICATION = new ArithmeticOperations("MULIPLICATION", 2);
    DIVISION = new ArithmeticOperations("DIVISION", 3);
    $VALUES = (new ArithmeticOperations[] {
      ADDITION, SUBTRACTION, MULIPLICATION, DIVISION
```

```
        });
    }
  }
```