OOPJ Notes Day-13 Session-1 Date: 11-05-2023

Working with Strings

- In C++ string s="DAC" //{'D','A','C','\0'} every string is ended with '\0' character
- In Java, String is collection of character instances which do not end with '\0' character.
- If we want to manipulate String in Java then we can use following classes:
  - java.lang.String
  - java.lang.StringBuffer
  - java.lang.StringBuilder
  - java.util.StringTokenizer
  - java.text.Pattern
  - java.text.Matcher
  - org.apache.commons.lang3.StringUtils #### String
- String is final class declared in java.lang package.
- It is sub class of java.lang.Object class and it implements Serializable, Comparable, CharSequence
- In Java, String is not primitive data type. Since it is a class, it is considered as non primitive / reference type.
- Even though String is non primitive type, we can create its instance with and without new operator.
- If we create String instance using new operator then it gets space on heap section.
- If we create String without new operator the it gets space on String literal on method area.
- In Java, String do not ends with '\0' character. Using illegal index, if we try to access character from String then String methods throws StringIndexOutofBoundsException.
- The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings.
  - Using + operator, we can contact any(primitive/non primitive) value to the String but using concat method we can concat onluy String.
- Constructor Summary of String class
  - public String()
  - public String(byte[] bytes)
  - public String(char[] value)
  - public String(String original)
  - public String(StringBuffer buffer)
  - public String(StringBuilder builder)
- Method Summary of String class:
  - public char charAt(int index)
  - public String concat(String str)
  - public boolean contains(CharSequence s)
  - public boolean equalsIgnoreCase(String anotherString)

  - public static String format(String format, Object… args)
  - public byte[] getBytes()
  - public int indexOf(int ch)
  - public int lastIndexOf(int ch)
  - public int indexOf(String str)
  - public int lastIndexOf(String str)
  - public String intern()
  - public int length()
  - public boolean matches(String regex)
  - public String[] split(String regex)
  - public boolean startsWith(String prefix)
  - public boolean endsWith(String suffix)
  - public char[] toCharArray()
  - public String toUpperCase()
  - public String toLowerCase()
  - public String trim()
  - public static String valueOf(Object obj)
- String twisters
- A Strategy for Defining Immutable Objects
  - Don't provide "setter" methods — methods that modify fields or objects referred to by fields.
  - Make all fields final and private.
  - Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final.
  - If the instance fields include re.ferences to mutable objects, don't allow those objects to be changed
  - Reference: https://docs.oracle.com/javase/tutorial/essential/concurrency/imstrat.html #### String Buffer vs String Builder
- StringBuffer and StringBuilder are final classes declared in java.lang packages.
- StringBuffer and StringBuilder are used to create mutable String instances.
- StringBuffer and StringBuilder class do not override equals and hashCode() method.
- To create instance of StringBuffer and StringBuilder, we must use new operator.
- StringBuffer is thread-safe/synchronized but StringBuilder is non thread-safe / unsynchronized.
- Since StringBuffer is synchronized, it is slower in performance. Since StringBuilder is unsynchronized, it is faster in performance. #### StringTokenizer
- Enumeration is interface declared in java.util package
  - boolean hasMoreElements()
  - E nextElement()
- StringTokenizer is a class declared in java.util package.
- The string tokenizer class allows an application to break a string into

tokens.

- Constructor Summary
  - public StringTokenizer(String str)
  - public StringTokenizer(String str, String delim)
  - public StringTokenizer(String str, String delim, boolean returnDelims)
- Method Summary
  - public int countTokens()
  - public boolean hasMoreTokens()
  - public String nextToken()
  - public String nextToken() #### Pattern and Matcher
- If we want to validate string then we should use regular expression
- Pattern and Matcher are final classes declared in java.util.regex package.
- An instance of the Pattern class represents a regular expression and nstances of the Matcher class are used to match character sequences against a given pattern. ### Java Reflection API
- Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.
- java.lang.reflect is the package where the required classes for the reflection has been provided.
- Important classes
  - Class
  - Field
  - Method
  - Constructor
- Some important methods:
  - Class classObj = obj.getClass() for Getting Class object for the respective class.
  - Method[] methods = classObj.methods() for Getting array of Method objects for public methods.
  - Field[] fields = classObj.fields() for Getting array of Field objects for public fields.
  - Constructor constructor = classObj.getConstructor() for getting Getting constructor
  - Method method1 = classObj.getDeclaredMethod("myFun",int.class,double.class)for Getting Method object for a particular method of a class
  - Field field1 = classObj.getDeclaredField("empName") for Getting Field object for a particular field
  - method1.setAccessible(true) - Making method accessible irrespective of the associated access specifier.
  - field1.setAccessible(true) - Making field accessible irrespective of the associated access specifier.
  - method1.invoke(obj,5,10.5) - Invoking the method with required parameter values
    - Note: static methods can also be invoked either by using classObj or obj.

– field1.set(obj,“Shyam”)- Setting the field value

```
import java.lang.reflect.*;
class MyClass {
  public int a;
  private int b;
  static int c;

  public MyClass() {
      System.out.println("No argument constructor invoked");
  }

  public MyClass(int x) {
      System.out.println("One argument constructor invoked");
  }

  public MyClass(int x, int y) {
      System.out.println("Two arguments constructor invoked");
  }

  public void myFun(int a, double b) {
      System.out.println("fun1 invoked");
  }

  public void myFun2() {
      System.out.println("myfun2 invoked");
  }

  private void myFun3() {
      System.out.println("myfun3 invoked");
  }

  private static void myFun4() {
      System.out.println("myfun4 invoked");
  }
}
public class RefComDemo {
  public static void main(String args[])
          throws NoSuchMethodException, IllegalAccessException, InvocationTargetExcepti
      MyClass myObj = new MyClass(5, 10);
      Class classObj = myObj.getClass();   //To Get class of Object
      Method[] methods = classObj.getMethods(); //To get methods of class
      Field[] fields = classObj.getFields(); //To get all feilds
      for (Method m : methods) {
          System.out.println(m.getName());
      }
      for (Field f : fields) {
```

```
            System.out.println(f.getName());
        }
        Constructor constructor = classObj.getConstructor(int.class);
        System.out.println(constructor.getName());
        Method method1 = classObj.getDeclaredMethod("myFun3");
        method1.setAccessible(true);
        method1.invoke(myObj);
        Method method2 = classObj.getDeclaredMethod("myFun4");
        method2.setAccessible(true);
        method2.invoke(classObj);
    }
}
```

JDBC Connectivity Example