Agenda

- Exception Chaning
- Custom Exceptions
- Date/LocalDate/Calender
- clone()
- Strings
    - String
    - StringBuffer
    - StringBuilder
- Garbage Collector
- Java BuzzWords
- ~~Enum~~
- ~~JVM Architecture~~
- ~~Arrays class~~

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.

Date/ LocalDate/ Calender

- Date and Calender class are in java.util package
- The class Date represents a specific instant in time, with millisecond precision.
- the formatting and parsing of date strings were not standardized it is not recommended to use
- As of JDK 1.1, the Calendar class should be used to convert between dates and time fields and the DateFormat class should be used to format and parse date strings.
- LocalDate is in java.time Package
- It is immutable and threadsafe class.

Java DateTime APIs

- DateTime APIs till Java 7

```
// java.util.Date
Date d = new Date();
System.out.println("Timestamp: " + d.getTime());
// number of milliseconds since 1-1-1970 00:00.
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
System.out.println("Date: " + sdf.format(d));

// java.util.Date
String str = "28-09-1983";
SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
Date d = sdf.parse(str);
System.out.println(d.toString());

// java.util.Calendar
Calendar c = Calendar.getInstance();
System.out.println(c.toString());
System.out.println("Current Year: " + calendar.get(Calendar.YEAR));
System.out.println("Current Month: " + calendar.get(Calendar.MONTH));
System.out.println("Current Date: " + calendar.get(Calendar.DATE));
```

- Limitations of existing DateTime APIs
    - Thread safety
    - API design and ease of understanding
    - ZonedDate and Time
- Most commonly used java 8 onwards new classes are LocalDate, LocalTime and LocalDateTime.
- LocalDate

```
LocalDate localDate = LocalDate.now();
LocalDate tomorrow = localDate.plusDays(1);
```

```
DayOfWeek day = tomorrow.getDayOfWeek();
int date = tomorrow.getDayOfMonth();
System.out.println("Date: " +
tomorrow.format(DateTimeFormatter.ofPattern("dd-MMM-yyyy")));

//LocalDate date = LocalDate.of(1983, 09, 28);
LocalDate date = LocalDate.parse("1983-09-28");
System.out.println("Is Leap Year: " + date.isLeapYear());
```

- LocalTime

```
LocalTime now = LocalTime.now();
LocalTime nextHour = now.plus(1, ChronoUnit.HOURS);
System.out.println("Hour: " + nextHour.getHour());
System.out.println("Time: " +
nextHour.format(DateTimeFormatter.ofPattern("HH:mm")));
```

- LocalDateTime

```
LocalDateTime now = LocalDateTime.now();
LocalDateTime dt = LocalDateTime.parse("2000-01-30T06:30:00");
dt.minusHours(2);
System.out.println(dt.toString());
```

Strings

- java.lang.Character is wrapper class that represents char.
- In Java, each char is 2 bytes because it follows unicode encoding.
- String is sequence of characters.
    - 1. java.lang.String: "Immutable" character sequence
    - 2. java.lang.StringBuffer: Mutable character sequence (Thread-safe)
    - 3. java.lang.StringBuilder: Mutable character sequence (Not Thread-safe)
- String helpers
    - 1. java.util.StringTokenizer: Helper class to split strings

String Class Object

- java.lang.String is class and strings in java are objects.
- String constants/literals are stored in string pool.
- String objects created using "new" operator are allocated on heap.
- In java, String is immutable. If try to modify, it creates a new String object on heap.

```
String name = "sunbeam"; // goes in string pool

String name2 = new String("Sunbeam"); // goes on heap
```

- Since strings are immutable, string constants are not allocated multiple times.
- String constants/literals are stored in string pool. Multiple references may refer the same object in the pool.
- String pool is also called as String literal pool or String constant pool.

StringBuffer and StringBuilder

- StringBuffer and StringBuilder are final classes declared in java.lang package.
- It is used create to mutable string instance.
- equals() and hashCode() method is not overridden inside it.
- Can create instances of these classes using new operator only. Objects are created on heap.
- StringBuffer implementation is thread safe while StringBuilder is not thread-safe.
- StringBuilder is introduced in Java 5.0 for better performance in single threaded applications.

String Tokenizer

- Used to break a string into multiple tokens - like split() method.
- Methods of java.util.StringTokenizer
  - boolean hasMoreTokens()
  - String nextToken()
  - String nextToken(String delim)

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.

Garbage Collector

- Garbage collection is automatic memory management by JVM.
- If a Java object is unreachable (i.e. not accessible through any reference), then it is automatically released by the garbage collector.
- An object become eligible for GC in one of the following cases:

```
// 1. Nullify the reference.
MyClass obj = new MyClass();
obj = null;

//2. Reassign the reference.
MyClass obj = new MyClass();
obj = new MyClass();

//3. Object created locally in method.
void method() {
MyClass obj = new MyClass();
// ...
}

- GC is a background thread in JVM that runs periodically and reclaim memory of unreferenced objects.
- Before object is destroyed, its finalize() method is invoked (if present).
- One should override this method if object holds any resource to be released explicitly e.g. file close, database connection, etc.

```JAVA
class Test {
    Scanner sc = new Scanner(System.in);

    @Override
    protected void finalize() throws Throwable {
        sc.close();
    }
}

public class Program{

    public static void main(String[] args) {
        Test t1 = new Test();
        t1 = null;
        System.gc();// request GC
    }
}
```

- GC can be requested (not forced) by one of the following.

    1. System.gc();
    2. Runtime.getRuntime().gc();

- GC is of two types i.e. Minor and Major.
```

1. Minor GC: Unreferenced objects from young generation are reclaimed. Objects not reclaimed here are moved to old/permanent generation.
2. Major GC: Unreferenced objects from all generations are reclaimed. This is unefficient (slower process).

- JVM GC internally use Mark and Compact algorithm.
- GC Internals: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Java BuzzWords

- 1. Simple
  - Simple for Professional Programmers if aware about OOP.
  - It removed the complicated fetaures like pointers and rarely used features like operator overloading from c++
  - It was simple till java 1.4
  - the new features added made it powerful (but also complex)
- 2. Object Oriented
  - Java is a object-oriented programming language.
  - It supports all the pillars of OOP
- 3. Distributed
  - Java is designed to create distributed applications on networks.
  - Java applications can access remote objects on the Internet as easily as they can do in the local system.
  - Java enables multiple programmers at multiple remote locations to collaborate and work together on a single project.
- 4. Compiled and Interpreted
  - Usually, a computer language is either compiled or Interpreted.
  - Java combines both this approach and makes it a two-stage system.
  - Compiled: Java enables the creation of cross-platform programs by compiling them into an intermediate representation called Java Bytecode.
  - Interpreted: Bytecode is then interpreted, which generates machine code that can be directly executed by the machine/CPU.
- 5. Robust
  - It provides many features that make the program execute reliably in a variety of environments.
  - Java is a strictly typed language. It checks code both at compile time and runtime.
  - Java takes care of all memory management problems with garbage collection.
- 6. Secure
  - Java achieves this protection by confining a Java program to the Java execution environment and not allowing it to access other parts of the computer
- 7. Architecture Neutral
  - Java language and Java Virtual Machine helped in achieving the goal of WORA - Write Once Run Anywhere.
  - Java byte code is interpreted by JIT and convert into CPU machine code/native code.
  - So Java byte code can execute on any CPU architecture (on which JVM is available)
- 8. Portable
  - As java is Architecture Neutral it is portable.

- Java is portable because of the Java Virtual Machine (JVM).

9. High Performance
   - Java performance is high because of the use of bytecode.
   - The bytecode was used so that it can be efficiently translated into native machine code by JIT compiler (in JVM).
10. Multithreaded
    - Multithreaded Programs handled multiple tasks simultaneously (within a process)
    - Java supports multi-process/thread communication and synchronization.
    - When Java application executes 2 threads are started
      1. main thread
      2. garbage collector thread.
11. Dynamic
    - Java is capable of linking in new class libraries, methods, and objects.
    - Java classes has run-time type information that is used to verify and resolve accesses to objects/members at runtime.