# Agenda

- Object class
- Methods of object class
- Abstract class/method
- JVM architecture
- Garbage collector
- Date/LocalDate/Calender
- ~~Enum~~
- ~~Interfaces~~
- ~~Marker interfaces~~

# Object class

- Non final and non-abstract class declared in java.lang package.
- In java, all the classes (not interfaces) are directly or indirectly extended from Object class.
- In other words, Object class is ultimate base class/super class hierarchy.
- Object class is not inherited from any class or implement any interface.
- It has a default constructor. Object o = new Object();
- Object class methods (read docs)
    - public Object();
    - public native int hashCode();
    - public boolean equals(Object);
    - protected native Object clone() throws CloneNotSupportedException;
    - public String toString();
    - protected void finalize() throws Throwable;
    - public final native Class<?> getClass();
    - public final native void notify();
    - public final native void notifyAll();
    - public final void wait() throws InterruptedException;
    - public final native void wait(long) throws InterruptedException;
    - public final void wait(long, int) throws InterruptedException;

# toString() method

- it is a non final method of object class
- To return state of Java instance in String form, programmer should override toString() method.
- The result in toString() method should be a concise, informative, and human-readable.
- It is recommended that all subclasses override this method.

# equals() method

- It is non final method of object class

- To compare the object contents/state, programmer should override equals() method.
- This equals() must have following properties:
    - Reflexive: for any non-null reference value x, x.equals(x) should return true.
    - Symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
    - Transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
    - Consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value x, x.equals(null) should return false.
- It is recommended to override hashcode method along when equals method is overriden.

## Abstract Methods

- If implementation of a method in super-class is not possible/incomplete, then method is declared as abstract.
- Abstract method does not have definition/implementation.
- If class contains one or more abstract methods, then class must be declared as abstract. Otherwise compiler raise an error.
- The super-class abstract methods must be overridden in sub-class; otherwise sub-class should also be marked abstract.
- The abstract methods are forced to be implemented in sub-class. It ensures that sub-class will have corresponding functionality.
- The abstract method cannot be private, final, or static.
- Example: abstract methods declared in Number class are:
    - abstract int intValue();
    - abstract float floatValue();

## Abstract class

- If implementation of a class is logically incomplete, then the class should be declared abstract.
- If class contains one or more abstract methods, then class must be declared as abstract.
- An abstract class can have zero or more abstract methods.
- Abstract class object cannot be created; however its reference can be created.
- Abstract class can have fields, methods, and constructor.
- Its constructor is called when sub-class object is created and initializes its (abstract class) fields.
- Example:
- java.lang.Number
- java.lang.Enum

## 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.

```
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.

# JVM Archicecture

- 1. Compilation
  - .class file is cretaed which consists of byte code
- 2. Byte Code
  - It is a machine level instructions that gets executed by the JVM
  - JVM converts byte code into target machine/native code
- 3. Execution
  - java is a tool used to execute the .class file.
  - It loads the .class file and invokes jvm for executing the file from the classpath
- JVM Archiceture Overview
  - ClassLoader + Memory Area + Execution Engine

# ClassLoader SubSystem

- It loads and initialize the class

## 1. Loading

- Three types of classLoaders
  - 1. BootStrap classloader that loads built in java classes from jre/lib jars (rt.jar)
  - 2. Extension classloader that loads the extended classes from jre/lib/ext directory
  - 3. Application classloader that loads the classes from the application classpath
- It reads the classes from the disk and loads into JVM method(memory) area

## 2. Linking

- Three steps
  - 1. Verifiaction : Bytecode verifier ensures that class is compiled by valid compiler and not tampered
  - 2. Preparation : Memory is allocated for static members and initialized with default values
  - 3. Resolution : Symbolic references in constant pool are replaced by the direct references

## 3. Initialization

- All static variables of class are assigned with their assigned values(field initializers)
- all static blocks are executed if present

# Memory Areas

- Their are 5 memory areas
  - 1. Method Area
  - 2. heap Area
  - 3. Stack Area

    - 4. PC Registers
    - 5. Native Method Stack Area

### 1. Method Area

- Create during JVM startup
- shared by all the threads
- class contents (for all classes) loaded into this area
- Method area also holds constant pool for all loaded classes.

### 2. Heap Area

- Create during JVM startup
- shared by all the threads
- All allocated objects (with new) are stored in heap
- The string pool is part of heap Area.
- The class Metadata is stored in a java.lang.Class object (in heap) once class is loaded.

### 3. Stack Area

- Separate stack is created for each thread in JVM (when thread is created).
- When a method is called a new FAR (stack frame) is created on its stack.
- Each stack frame conatins local variable array, operand stack, and other frame data.
- When method returns, the stack frame is destroyed.

### 4. PC Registers

- Separate PC register is created for each thread.
- It maintains address of the next instruction executed by the thread.
- After an instruction is completed, the address in PC is auto-incremented.

### 5. Native Method Stack

- Separate native method stack is created for each thread in JVM (when thread is created).
- When a native method is called from the stack, a stack frame is created on its stack.

## Execution Engine

- The main component of JVM
- Convert byte code into machine code and execute it (instruction by instruction).
- It consists of
    - 1. Interpreter
    - 2. JIT Compiler
    - 3. Garbage Collector

### 1. Interpreter

- Each method is interpreted by the interpreter at least once.
- If method is called frequently, interpreting it each time slow down the execution of the program.
- This limitation is overcomed by JIT (added in Java 1.1).

### 2. JIT compiler

- JIT stands for Just In Time compiler.
- Primary purpose of the JIT compiler to improve the performance.
- If a method is getting invoked multile times, the JIT compiler convert it into native code and cache it.
- If the method is called next time, its cached native code is used to speedup execution process.

### 3. Profiler

- Tracks resource (memory, threads, ...) utilization for execution.
- Part of JIT that identifies hotspots. It counts number of times any method is executing.
- If the number is more than a threshold value, it is considered as hotspot.

### 4. Garbage Collector

- When any object is unreferenced, the GC release its memory.

## JNI

JNI acts as a bridge between Java method calls and native method implementations.

## 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.