JAVA HEADLINES from SARAVANA

History
- project started on 1991
- James Gosling, Mike Sheridan, Patrick Naughton
- v1.0 -> 1996
- 2006/07 -> JVM -> FOSS (Free & Open Source Software)
- 2009/10 -> Oracle acquired Sun Microsystems and become owner of Java
- latest version 15

Design Goals
- simple, object oriented, familiar
- robust and secure
- architectural neutral and portable
- high performance (JIT)
- interpreted, threaded and dynamic

Release
- v1.0 -> 1996
- v1.1 -> 1997
- v1.2 -> 1998 => J2SE, J2EE, J2ME
- v1.3 -> 2000
- v1.4 -> 2002
- v5.0 -> 2004=> JSE, JEE, JME
- v6.0 -> 2006
- v7.0 -> 2011
- v8.0 -> 2014 (LTS) => OOP + FP (Lambda Expr + Stream API)
- v9.0 -> 2017
- v10 -> 2018(Mar)
- v11 -> 2018(Sep) (LTS)
- v12 -> 2019(Mar)
- v13 -> 2019(Sep)
- v14 -> 2020(Mar)
- v15 -> 2020(Sep)

JDK vs JRE vs JVM

Java Editions
- Java SE (Standard Edition)
- Java EE (Enterprise Edition)
- Java ME (Micro Edition)

Language Constructs

- Data Types
    - byte   (1 byte)
    - short   (2 bytes)
    - int (4 bytes)
    - long   (8 bytes)
    - float   (4 bytes)
    - double (8 bytes)
    - boolean (1 byte)
    - char   (1 bytes)

    - string

- Variables
    - Primitive - int, float, double, char, boolean, etc...
    - Non-Primitive (Reference) - String, Array, Object

- Constants
    - final int ID;

- Operators
    - Arithmetic Operators
    - Relational Operaytors
    - Bitwise Operators
    - Assignment Operators
    - Conditional Operators
    - instanceof operator

- Expressions, Statements, Blocks
    - statements should end with semicolon ;
    - blocks will be created with set of {}

- Control Flow Statements
    - IF ELSE
    - Switch

- Loop Statements
    - while
    - do...while
    - for
    - foreach

    while(true) {
    }

for(int i =0; i < 10; i++) {}

for(;;){}

for(String item: items) {}

- Branching Statements
- Naming Conventions
    - starts with $, _ and alphabets
    - can contain numbers

- Comments
    - single line => //
    - multi line => /* */
    - class comment => /** */

- Arrays
    - int[] ids;
    - int ids[];

    - int[] ids = {10,20,30}

    - String[] ids = new String[10];
        ids[0] = 10;
        ids[1] = 20;
        ids[2] = 30;

- Strings
    - String name = new String("Lowes");
    - String company = new String("Lowes")

    - String name = "Lowes" + "Inc"
    - String company = "Lowes"

    - String Buffer vs String Builder

    - StringBuffer name = new StringBuffer("Lowes")
    name.toString()

    - company = new StringBuffer("Lowes")

Object Oriented Programming

- Class
    - Template or Blueprint

- Object
    - Instance of a class

- Abstraction
    - showing up only essentials details
    - abstract class
    - can't instantiate
    - extending class should provide implementation for all abstract methods

- Encapsulation
    - data hiding
    - access modifiers
    - default, private, protected, public

- Inheritance
    - acquiring/inheriting state and behavior from parent class
    - multi level inheritance supported
    - using extends
    - multiple inheritance is not supported due to ambiguity during run time
    - with interfaces can support multiple inheritance

- Polymorphism
    - ability to take multiple forms
    - overloading - compile time (static) polymorphism
    - overriding - runtime polymorphism

- Interface
    - contract b/w class and outside world
    - contain only abstract methods
    - constants
    - using implements
    - marker interfaces - no method declarations

- Object
    - base/root class
    - hashCode()
    - equals()
    - toString()

Collections

- An object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.
- A collections framework is a unified architecture for representing and manipulating collections

[Collection Framework]
- Interfaces - Collection, Set, List, Queue, Deque, Map
- Implementations - ArrayList, LinkedList, HashSet, TreeSet, LinkedHashSet, ArrayDeque, HashMap, TreeMap, LinkedHashMap
- Algorithms - Sorting, Shuffling, Searching, Data Manipulation, Composition, Min/Max

[Benefits]
- Reduces Programming Effort
- Increases Program Speed and Quality
- Allows interoperability among unrelated APIs
- Reduces effort to learn and to use new APIs
- Reduces effort to design new APIs
- Fosters software reuse

Arrays
- Fixed size
- Sequential memory allocation

Vector
- Dynamic Array
- Synchronized. Thread Safe

Hashtable
- Key Value store (Objects)
- Synchronized. Thread Safe

Properties
- Key Value store (String)

List - indexed and ordered
- Vector
- ArrayList
- LinkedList

Set - maintains unique values and sorted
- TreeSet
- HashSet
- LinkedHashSet

Map - key value store
- TreeMap
- HashMap
- LinkedHashMap

Queue - FIFO
- PriorityQueue
- Deque

Collections Framework Hierachy

- Iterable
  - Collection
    - List - dynamic sizing, ordered, index based, supports
      - Vector - synchronized
      - ArrayList - faster frequent reads, random access
      - LinkedList - frequent insertions and updations
    - Set - not allows duplicates, sorted, no index
      - HashSet - faster search, works based on hashing techinque, not ordered/sorted, allows null values
      - TreeSet - sorting, not allows null values
        - Comparable
          - compareTo(object ob)
        - Comparator
          - compare(Object ob1, Objec ob2)
      - LinkedHashSet - ordered, maintains insertion order

    - Queue - FIFO
      -Deque
        - PriorityQueue
        - BlockingQueue

- Map - maintains data as key, value pair, not allows duplicate keys
  - Hashtable - not allows null keys, values
  - HashMap - allows null keys and values
  - TreeMap - not allows null keys and allows null values
  - LinkedHashMap - allows null keys and values

Generics
- Stronger type checks at compile time
- Elimination of casts
- Enabling programmers to implement generic algorithms
- Type Parameter Naming Convention

E - Element (used extensively by the Java Collections Framework)
K - Key
N - Number
T - Type
V - Value
S,U,V etc. - 2nd, 3rd, 4th types

Reflection

- An API that represents ("reflects") the classes, interfaces, and objects in the current Java Virtual Machine.
- Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine

[Use cases]
- Extensibility Features
- Class Browsers and Visual Development Environments
- Debuggers and Test Tools

[Limitations]
- Performance Overhead
- Security Restrictions
- Exposure of Internals

Annotations

-Annotations, a form of metadata, provide data about a program that is not part of the program itself

[Use cases]
- Information for the compiler
- Compile-time and deployment-time processing
- Runtime Processing

[Predefined Annotations]
- @Deprecated
- @Override
- @SuppressWarnings
- @SafeVarArgs
- @FunctionalInterface

[Custom Annotations]

[Meta Annotations]
- @Documented
- @Inherited
- @Retention
- @Target
- @Repeatable

Nested/Inner Classes

- A nested class is a member of its enclosing class.
- Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.
- Static nested classes do not have access to other members of the enclosing class

[Why Nested Classes]

- It is a way of logically grouping classes that are only used in one place
- It increases encapsulation
- It can lead to more readable and maintainable code

[Types]

- Static Nested Classes
- Inner Classes (Non-static)
    - Local Inner Class -> declare an inner class within the body of a method
    - Anonymous Inner Class -> declare an inner class within the body of a method without naming the class

Functional Programming

- data immutable
- computations are done by evaluation of functions
- returning values instead modifying
- functions can be passed as arguments
- assign functions to variables

Lambda Expressions

- Lambda expressions enables to treat functionality as method argument, or code as data.
- Functional Interface
    - performs single task
    - has only one abstract method
- Consumer - One argument, no return values
- Supplier - No argument and only return a value
- Function - Take an argument and return a value
- Predicate - Take an argument and always return boolean

Stream APIs

- Streaming Source
- Intermediate Operations -> filter, map, sort, distinct, etc..
- Termial Operations
    - Reduce Operation -> count, avg, sum, min/max, etc..
    - Collect Operation -> toList, toSet, grouping, etc..

Other Java 8 Features
- Default Methods
- Optional
- New DateTime API

Multi-threading and Concurrency API
- Concurrent Programming Overview
- Process vs Threads
- Thread Creation
    - Extend Thread
    - Implement Runnable
    - Implement Callable

- Thread Lifecycle
    - New -> Runnable -> Running -> Sleep -> Runnable
    - New -> Runnable -> Running -> Waiting -> Notify -> Runnable
    - New -> Runnable -> Running -> Blocked -> Acquire Lock -> Runnable
    - New -> Runnable -> Running -> Dead

- Thread Constructs
    - sleep - holds lock and put the thread to sleep state until the time elapses
    - wait - release locks and put the thread to wait state until receive notification
    - notify - sends notification to the waiting threads
    - notifyAll - sends notification to all the waiting threads
    - join - makes parent thread to wait until child thread terminates
    - interrupt - interrupts the thread execution
    - yield - gives the chance for other threads to runs
    - volatile - transient shared variable
    - synchronized <- Thread Interference, Memory Inconsistency
    - Thread Synchronization
    - Thread Liveness
        - Starvation
        - DeadLock
    - Concurrency API
        - Callable
        - Executors
        - Thread Pool
        - Future

        - ForkJoin Framework
        - CompleteableFuture

- Concurrent Collections
- Locks and Conditions
- Atomic

Java I/O

- An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays
- Byte Stream -> Programs use byte streams to perform input and output of 8-bit bytes
- Character Stream -> Java platform stores character values using Unicode conventions. Character stream I/O automatically translates this internal format to and from the local character set.
- Buffered Stream -> Buffered input streams read data from a memory area known as a buffer; the native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full.
    - Flushing Buffer Stream to sync write with native file sys
- Scanning and Formatting
- I/O from command line

- File I/O
    ByteStream
        - FileInputStream
        - FileOutputStram
    CharacterStream
        - FileReader
        - FilerWriter
    BufferedStream
        - BufferedReader
        - BufferedWriter
        - BufferedInputStream
        - BufferedOutputStream

- Console I/O
    Scanner
    Console
    BufferedReader

- Serialization / De-serialization
    ObjectOutputStream
    ObjectInputStream

JDBC API
- Specification to access RDBMS from Java
- 4 Types of drivers
- Steps required to access database
    - Registering database drivers OR creating datasource
    - Get Connection
    - Create Statement
    - Execute Query
    - Access ResultSet
- RowSet provides more features to access the data on top of Resultset

Reference: https://docs.oracle.com/javase/tutorial/

[Appendix]
Class Libraries
- Generics
- Collection Libraries
- Functional Programming
- Concurrency
- IO/NIO
- Networking
- Reflection
- XML Processing
- Security
- Internalization and Localization

- JDBC
- JNDI
- RMI and CORBA
- JMX

- AWT
- Swing
- JavaFX

Java Tools
[Basic Tools]
- java -> launcher for Java applications
- javac -> compiler for the Java programming language
- javadoc -> API documentation generator
- jar -> create and manage Java Archive (JAR) files
- javap -> class file disassembler
- jdeps -> java class dependency analyzer

[Monitoring & Management]

- jcmd -> JVM Diagnostic Commands tool - Sends diagnostic command requests to a running Java Virtual Machine
- jconsole -> graphical tool for monitoring a Java virtual machine
- visualvm -> provides memory and CPU profiling, heap dump analysis, memory leak detection, access to MBeans, and garbage collection
- jmc -> java mission control - includes tools to monitor and manage your Java application without introducing the performance overhead
- jps -> JVM Process Status Tool - Lists instrumented HotSpot Java virtual machines on a target system
- jstat -> JVM Statistics Monitoring Tool - Attaches to an instrumented HotSpot Java virtual machine and collects and logs performance statistics
- jinfo -> prints configuration information for a given process
- jmap -> prints shared object memory maps or heap memory details of a given process
- jhat -> starts a web server on a heap dump file (for example, produced by jmap -dump), allowing the heap to be browsed
- jstack -> prints a stack trace of threads for a given process

[Scripting]
- jjs -> runs the Nashorn command-line script shell

Garbage Collectors
- Serial Garbage Collector [Single threaded. Freezes all app threads during GC]
- Parallel Garbage Collector [Multi threaded. Freezes all app threads during GC]
- Concurrent Mark Sweep [Multi threaded with shorter GC pauses]
- G1 Garbage Collector [Divides heap space into many regions and GCs region have more garbage]