Java Core

Java Interview Guide : 200+ Questions

Experimental Java Based JIT Compiler

- Graal
  - Purpose to improve performance
  - To Use
    - -XX:+UnlockExperimentalVMOptions
    - XX:+UseJVMCICompiler
- Other Java 10 Features

Garbage Collector Interface	Interface improves the source code for isolation fo diferent garbage collectors
Parallel Full GC for G1	This feature improves G1 worst-case latencies by making the ful GC-parallel
Heap Allocation on Alternate Memory Devices	Enable HotSpot VM to allocate heap on alternate devices specified by the user
java will provide the task of javah which was removed	

Java ClassLoaders

- When a program is executed, JVM needs to load the content of all the needed classes
  - Uses a class loader to find the classes
- Three Class Loaders (Based on where they would search for a class)
  - System Class Loader → Loads all class from Classpath
    - The programmer writes these classes
    - Search the classpath
    - Load class, java, war and ear
    - Load files in rt.jar and i18n.jar
  - Extension Class Loader → Loads all classes from extension directory for platform specific functions
    - Extension Directories : jre, ext, lib
    - Search inside the latter directories
    - Platform specific
  - Bootstrap Class Loader → Load all the java core files
    - Load all the java core files
  - Order : System ClassLoader, ExtensionClassLoader, BootStrap ClassLoader and then ClassNotFound Exception

Java 11 Removed Features

- Features Removed
  - Modules
    - java.corbra
    - jdk.snpm
    - java.xml.ws
    - java.xml.bind
    - JavaFx
  - Classes
    - `com.sun.awt.AWTUtilities sun.misc.Unsafe.defineClass
  - Methods
    - Thread.destroy() Thread.stiop(throwable)
  - Tools
    - appletview Launcher
    - wsimport
    - wsgen
    - scheamgen
    - Xjc
  -

Wrapper Classes

- Reason for Wrapper Classes – A wrapper around a primitive datatype (ex. int, float)
  - Null is a possible value
  - Use it in a Collection
  - Methods that support Object Like Creation from other types.
- valueOf → A chance you are using cached value
- Wrapper Classes are immutable
- Autoboxing → the automatic conversion the compiler doe between the primitive types and their corresponding object wrapper class
  - AutoBoxing helps in saving memory by reusing already created Wrapper Objects
  - AutoBoxing uses valueOf Methods
- Creating object with new → The object are different
- Casting
  - Convert one datatype into another
  - Implicit Casting → Automatically done by Compiler
    - Ex. automatic widening conversions (smaller values in larger variable type)
  - Explicit Casing
    - Narrowing conversions (storing larger values into smaller variable types;
    - Cause the truncation of value if the value stored is greater than the size of a variable
- Why would it be more secure to store sensitive data in a character array rather than a string
  - String are immutable, so they stay in memory until garbage collected.
  - With mutable object you can change and the password will no longer be in memory.
-

Optional

- A container for an object which may or may not contain a non-null value
  - empty   // Returns a empty instance of the Optional
  - filter
  - get() –> Not found throws NoSuchElement, usually know there is one entry
  - ifPresent → call the function
  - orElse → Returns the Optional describing the specified value present non null value
  - orElseGet(supplier) returns value if present or calls function

Java 10 – Type Inference with var

- Type inference with var
  - infer data type based on value
  - var must be initialized or an error will be given
  - var cannot be initialized to null or an error will be given
  - Example
    - var dateOfBirth2 = new Date()   // DataType is Date
    - var salary2 = 5.0;   var salary3 = sallary2 * 2;   // Result is 10 and type is float
- Quiz Question
  - Code #1   HashMap<Integer, String> studentMap = new HashMap<Integer, String>();
  - Code #2   var studentMap = new HashMap<>();
  - Can we achieve similar functionality from Code#1 and Code#2 → Yes

String , StringBuffer, StringBuilder

- Immutable : Once you created an object its value cannot be changed.
- Where are string values stored in memory
  - Example String str = “value”
    - Stored in a String Constant Pool inside the Heap Memory.
    - If it exist it is reused
  - Example String str = new String(“test”);’
    - The object is created on the heap
    - No Reuse
- Don’t use Strings Concatenation in loops
  - Each time a Concat is done a new object is created.
- StringBuffer : ThreadSafe, mutable
- StringBuilder : Not Thread Safe, mutable
- String : Thread Safe, Immutable

String Handling in Java 11

- New Functions
  - repeat(100) Parameter : Number of times to repeat
  - isBlank (t) – The String contains no characters
  - lines When a String contains multiple lines then it will print each line out on a seperateline
  - strip Remove the spaces from the left an right
  - stripLeading Remove the white space from the Left
  - stripTrailing Remove the white space from the right
- Number Formatting in Java 12
  - Number fmt = NumberFormat.getCompactNumberInstance();
  - fmt.format(number=1000); // The result would have been 1000, but will now be 1k
  - number can also be equal to 1 million
  - There are two styles : NumberFormat.Style.LONG and NumberFormat.Style.SHORT
- String Methods in Java 12
  - indent() Adds spaces to the left of the a String. Has a parameter for the number of spaces
  - transform() requires a Function Parameter that expects a function which expects a String and Returns a Result
    - example “Java”.transform(str → str + “Fast Forward”)

Equals and HashCode Methods

- == : checks the object references
- equals : Check the object state is the same for both.
  - If the class or parent classes do not define equals then the equals from the Object class is used which checks the references
- Example : Over the implementation
  - Public boolean equals (Object obj) {
    - Client other = (Client)obj;
    - If (id != other.id) { Return false; } else return true; }}
- Equals should satisfy the following properties
  - Reflexive
  - Symmetric
  - Transitive
  - Consistent
    - For any reference value x,y and z : if x.equals(y) and y.equals(z) returns true then x.equals(z) must return true
  - For any non null value x.equals(null) must return false;
- Good idea to check for the class in the equals method
- HashCode()
  - Used to decide which group an object should be placed into
  - A good hashing function evenly distributes object's into different groups
  - If ojb1.equals(obj2) then obj1.hashCode() should be equal to obj2.hasCode()
  - Obj.hashCode() should return the same value when run multiple times
  - If two objects are not equal they might have the same hashcode

Equals and HashCode Methods

- Example
  - Public int hashCode() {
    - Final int prime = 31;
    - Int result = prime + id
    - Return result;
- When you overwrite equals overwrite hashcode
  - If two objects are equal according to the equals(Object) method, then calling the hashcode() method on each of the two objects must produce the same integer result.

Statements

- Switch Variables can be char, byte, short, int, enum or String
- For (;;) → compiles and creates an infinite loop
- Break → Breaks out of the inner most loop
- Java 12 --Experimental – Different format
  - switch(month) {
    - case January season = “Winter”;
    - case March, April, May:
    - season = “Spring”;
    - break;
    - }
  - Addressing problems with the Switch Statement
  - No longer need the break statement
- Example
  - enum Month { January, February, March }
  - string season = switch(Month):
    - case January, February, March → season =”Winter”;

Exception Handling, Try Catch

- In the exception include some sort of ID so the person can search the back end logs to find the actual exception
- Exceptions follow the Chain of Responsibility Pattern. The jvm check if it can handle exception and if not it throws it previous function until it reach then JVM then it used a standard exception (Message and Stack Trace)
- When is finally not executed
  - If the exception is thrown in finally
  - If JVM crashes in between
- Finally will be executed even if the try and catch block both have a return.
- Checked and Unchecked Exceptions
  - Throwable → Top Most Exception Hierarchy
  - Error → Any action that cannot be handled (extends throwable). It a serious problem that the program should not catch
    - JVM runs out of memory
  - Exception → Any action that can be handled (extends throwable)
    - File not found
    - A checked exception.
  - RuntimeException (Extends Exception)
  - CheckedException extends Exception – Know at compile time
    - Need the throws in the method signature or a try/catch or add the throws to the function signature (checked at compile time)
    - Examples SQLException, IOException
    - Also throw the exception
  - UnCheckedException extends RuntimeException == Known only at runtime
    - I am throwing the exception, but the calling method may not be able to do anything about it. : ArrayIndexOutOfBoundsException
    - Not checked at compilation , but happen at run time (associated with data)
    - Can still use the try catch block and throws

Custom Exceptions
Try with Resources

- Extends Exception → Create Checked Exception
- Extends RuntimeException → Creates Unchecked Exception
- A compiler error will happen if the specific exceptions (CurrenciesDoNotMatchException) catch blocks are not before the catch block for the generic exception (Exception)
- New feature in Java 7 → Handle multiple exceptions types in the same exception handling block
  - Try {
  - } catch (IOException | SQLException exception) {
  - }
  - Write less redundant code
- Try with Resources
  - Example
    - Try (BufferedReader br = new BufferedReader(new FileReader(“FILE_PATH”))) {
      - String line
      - While ((line = br.readline()) != null) {   System.out.println(line); }|
    - } catch (IOException e) { e.printStackTrace(); }
  - The class must implement the Autoclose interface
  - Ensures each resource is closed.
  - Fixes
    - Memory Leak since the resource may not be closed.
    - No check for null
    - No unnecessary finally block
  -

Try with Resource Enhancements

- Try with Resource Enhancement
  - Resource Can be declared outside try
  - Example
  - Connection conn = DriverManager.getConnection(url, user, password);
  - Statement stmt = conn.createStatement()
  - ResultSet rs = stmt.executeQuery(query)
  - try (conn; stmt; rs) {
    - // Database Code
  - }

Variable Arguments

- public int sum(int… numbers) {
  - Int sum = 0;
  - for (int number: numbers) {
      - sum += number;
      - }
      - return sum;
    - }
  - }
- The variable arguments should always be the last parameter of the method so the variable arguments can be zero and there can be argument of the same type before the variable arguments
  - Example
    - public int getData(int a, int b, int… restOfArgs)
      - here you can getData(1), getData(1,2), getData(1,2,4,5,6)   // Are all valid
    - public in getData (int.. restOfArgs, int a, int b)
      - would give a compilation error
      - If it did the output would be restOfArgs = 1,2,4 and a, b would have an unknown value
- Can also do it with classes too.
  - Example: void bark(Animal… animals) // Animal is a class

Enum

- Final class with a fixed number of instances
- Use inf implementing the strategy pattern
- Example
  - enum Season {
    - WINTER(1), SPRING(2), SUMMER(3), FALL(4);
    - Private int code; // If enum has a constructor defined a value can be defined
    - Public int getCode() { return code; } // can have this since we added int code
    - Public static SeasonCustomized valueOf(int code) {
      - For (SeasonCustomized season : SeasonCustomized.values()) {
        - If (season.getCode() == code) {
          - Return season; }}
          -
    - Public int getExpectedMaxTemperature() {
      - switch(this) {
        - Case Winter: return 5; break;
        - Case Spring return 5; break;
        - Case FALL: return 10; break;
        - Case Summer: return 10; break;
        - Default: return -1; }
    - SeasonCustomized(int code){ this.code = code; }
    - };
- Example → Extended the Enum Directly and extending the customized method
  - Enum SeasonCustomized {
    - Winter(1) { public int getExpectedMaxTermperature() { return 5; } };
- Can use a switch around the enum as of Java 7

Asserts and Garbage Collection

- Asserts – If false then throw an AssertionError
  - Error → Developer cannot handle it.
  - Example assert(principal > 100);
  - Check for cases that are not suppose to happen
- Garbage Collection
  - Automatic Memory management in Java
    - Keep as much of the heap free as possible
  - Removes objects that go out of scope and/or have a reference count of 0
  - When does Garbage Collection run
    - When the available memory on the heap is low
    - When the cpu is free
  - JVM throws an OutOfMemoryException when memory is full and no objects on the heap are eligible for garbage collection
  - finalize();
- G1 → Garbage First
  - Used by servers where there is a lot of memory and CPU
- Before Java 7
  - Divide heap into three regions and space is reserved
  - Young/Eden
  - Old
    - Permgen/Permanent
-

Asserts and Garbage Collection

- From Java 7
  - Garbage collector divides Java Heap Memory into multiple regions/blocks which are Young/Old/Survivor
  - No space is reserved for any region
  - The Size of each block/region is equal
    - The size of these blocks can be configured and varies between 1 and 32 MBExplicit Examples
- Garbage Collection Issues
  - Defining a class with a static field and not setting it to null when it needs to be released
  - Group of Objects with Circular Dependencies.
    - Garbage collectors can’t decide whether they are needed or not.
  - https://stackify.com/memory-leaks-java/
- Garbage Collection
  - Excessive Garbage Collection happen when there is a large amount of short lived objects
  - The garbage collector works excessively to remove those object
-

Static and Member Initializers

- Initialization Blocks
  - Static Initializer → Run when the class is loaded
    - Can only reference static variables
  - Instance Initializer → Run when a new object is created
    - Can reference only static variables and instance variables
- Example
  - Public class InitializerExamples {
    - Static int count;
    - Int count2;
    -
    - static {
      - System.out.println(“Static Initializer”);
      - System.out.println(“Count when Static Initializer is run is “ + count);
    - }
    -
    - {   // Instance Initializer
      - Count2 = 6;
    - }

JPMS

- Java Platform Module System
- Main Purpose is Modularity
- Before Java 9 java application development was based on jars.
- From Java 9 java application development was based on modules
- Use Case #1
  - Suppose you are creating a java application which requires only 5 java class from rt.jar and you are on a mobile device
  - The jar is 61k with Java 9 has been divided into different modules so can import only the modules needed therefore decrease the size

java.base.jmod	17kb
java.compiler.jmod	109kb
java.naming.jmod	678kb

- Use Case #2
  - Suppose you application contains 500 java classes
  - Before Java 9 1 jar file for 500 java classes
  - Suppose the third party want to use one Java Class
  - From Java 9 Divide the jar file into small modules

JPMS 2 – How to create modules

- Eclipse
  - When creating a new project for the source tab : create module-info.java file should be checked
- The file is called module-info.java – Found in the src directory
  - This files needs the module name
  - What does it export
  - What does it require
- How to compile
  - java -d mods --module-source-path src/ --module com.jff.mymodule
  - java --module-path mods/ --module com.jff.mymodule com.jff.mymodule
- Will need to provide a module name : com.jff.mymodule   follows the import function standard
  - example com.jff.mypackage
- module com.jff.mymodule {
  - require java.base
  - }
- In Eclipse fix project setup. Add module path to the modulePath

Java 9 – Jlink

- Java Linker-- Assemble and optimize a set of modules and their dependencies into a custom runtime image
- Create custome/JRE
- Use Case
  - Suppose your application contains only 5 java files
  - To run this small application we need to install complete JRE which contains hundreds of classes
    - size of JRE is about 200MB
    - Problems
      - Memory Wastage
      - Default JRE is not recommended for small devices or IOT devices until Java
  - jlink --module-path out --add-modules helloModule.java.base --output jretest2
    - cd jretest2
    - get bin, conf, include, legal lib
    - java -m helloModule/mypackage.HelloDemo

Annotation Definition

- Annotations
  - A form of metadata (provides data about a program that is not part of the program
  - Have no direct effect on the operation of the code they annotate
- Uses
  - Information for the compiler
  - Compile Time and Deployment Time Processing
    - Software tools can process annotation information to generate code, XML files and so forth
  - Runtime Processing
- Data about your code that does not affect your code, but is used by other programs/libraries
- Information about data
- Applies to class, fields, methods
- Example of annotation
  - The annotation tells the compiler that the warning for the depreciated class should not have a warning printed out fot itl.
  - @Supresswarnings(value=”deprecation”)
  - File file
-

Creating an annotation

- Uses
  - Compiler
  - @SupressWarnings
  - @Override → Causes an error if it is not overriding something.
  - @Deprecated
  - Testing code
    - JUnit
  - Generate Code
  - How the data is to be written to a database.
  - Runtime
    - Serialization
    - Ioc Dependency Injection
- Object Relation Model Dependency InjectionDeclare an annotation
  - @Retention(RetentionPolicy.RunTime) // Can have three values Source, Runtime, Class
  - Public @interface PrintingDevice
  - {
    - String defaultPrintMehthod()
    - int defaultNumberOfCopies()
  - }
- For a Class Field or Method you can add the annotation
  - @PrintingDevice(defaultPrintMethod = “print”, defaultNumberOfCopies = 5)
  - Class PrintingDevice {}
- Useful when the class have different signatures, but should have the same signature. An example would be a printer, disk, monitor.   All write output, but each has different functions.
-

Creating a Custom Annotation

- To Retrieve the annotations for a class
  - PrinterDeveice printDevice = <class name>.getClass.getAnnotation(“PrintingDevice.class”)
  - Method PrintMethod = PrintDevice.getClass().getMethod(“printDevice.defaultPrintMethod”, int.class);
    - 2nd parameter is the value of the parameter.
  - PrintMethod.invoke(printDevice, printDevice.defaultNumberOfCopies)

Tail Recursion

- Tail recursion happens when the recursive call in the tail position within the enclosing context
  - Public int sumnFromOneToN(int n, int a) {
    - If (n < 1) return a else Return sumFromOneToN(n – 1, a + n); }
  - Need to be aware of this limitation to avoid StackOverflowError
  - In Java Iterator is universally preferred to recursion