JAVA QC Questions

Java Basics

1. What is Java? / Why Java?

- A high-level OOP language with rich API libraries, widely used around the world, supported by Oracle. Write once run anywhere (WORA), static types, compiled language

2. What is JRE / JDK / JVM?

- JVM - Java virtual machine. Runs compiled Java code
- JRE - Java runtime environment. Contains the JVM.
- JDK - Java developer kit. Has a compiler, debugger, etc. Contains JRE and JVM.

3. What is an object / class?

- Class is a blueprint for an object

4. What is the root class from which every class extends?

- The Object class

5. What are the primitive data types in Java?

- boolean, byte, short, int, long, float, double, char

6. Where are Strings stored?

- In the String pool in the heap

7. Explain stack vs heap?

- Heap is where objects are stored in memory. Stack is where local variable references are kept - a new stack is created for each method invocation

8. What are annotations?

- A type of syntactic metadata added to the code, read by the compiler - use @ syntax

9. What is a POJO? What is a bean?

- POJO - plain old Java object. Any Java object that you create.

- Bean - a POJO that has private data members, public getters/setters, and overrides .hashcode, .equals, and .toString methods

10. How can you force garbage collection in Java?

- Garbage collection cannot be forced but only requested using `System.gc()`.

11. Why are strings immutable in java?

- Identical String literals are collected in the "String pool" in an effort to conserve memory. Reference variables will then point to the same String object instance. Changing the object's state in the String pool will make changes to all references to that String object. Instead, when a change to a String is made, the JVM makes a new String object, and the reference variable points to the new String in the String pool.

12. What is the difference between `String`, `StringBuilder`, and `StringBuffer`?

- Strings are immutable. Both `StringBuilder` and `StringBuffer` are mutable. Furthermore, `StringBuffer` is sychronized while `StringBuilder` is not.

13. What are the access modifiers in Java? Explain them.

- public - can be accessed from any package.
- private - only members of the same class can access.
- protected - can be accessed by classes inside the package and subclasses anywhere.
- default - no access by classes or subclasses outside the package

14. What are the non-access modifiers in Java?

- static, final, abstract, default, synchronized, transient, volatile, native, strictfp

15. What is the difference between static and final variables?

- Static variable is a global variable shared by all the instances of objects and it has only single copy. A final variable is a constant variable and it cannot be changed.

16. What are the default values for all data types in Java?

- Objects - null. int, short, byte, long, float, double - 0. boolean - false. char - 'u0000' (null character)

17. What is a wrapper class?

- Wrapper class is a wrapper around a primitive data type. It represents primitive data types in their corresponding class instances e.g. a boolean data type can be represented as a Boolean class instance. All of the primitive wrapper classes in Java are immutable i.e. once assigned a value to a wrapper class instance cannot be changed further.

18. What is autoboxing / unboxing?

- Auto-boxing is the automatic conversion of primitives to their wrapper classes by the compiler. Useful for adding primitives to collections

19. Is Java pass-by-value or pass-by-reference?

- Java is strictly pass by value. Even when object references are passed as arguments, it is the value of the reference that is passed

20. What makes a class immutable?

1. Declare the class as `final` so it can't be extended.
2. Make all fields private so that direct access is not allowed.
3. Don't provide setter methods for variables.
4. Make all mutable fields final so that it's value can be assigned only once.
5. Initialize all the fields via a constructor performing deep copy.
6. Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

21. If two objects are equal, do they have the same hashcode? If not equal?

- If two objects have the same hashcode then they are NOT necessarily equal. But if objects are equal, then they MUST have same hashcode.

22. What data types are supported in switch statements?

- `String`, `int`, `char`, `short`, `byte`, `enums`

23. List all non-access modifiers?

- `static`, `final`, `abstract`, `default`, `synchronized`, `transient`, `volatile`, `native`, `strictfp`

24. How to pass multiple values with a single parameter into a method?

- Use varargs

25. What is static block?

- Used for static initialization. Executed only once - upon creation of first object of class or access to static method of class

26. What is static imports?

- Importing a static method or variable from a class - syntax: import static

27. What methods are available in the Object class?

- `.clone`, `.hashcode`, `.equals`, `.toString`

28. How would you clone an object?

- First, tag the class with the `Cloneable` marker interface. Next, invoke `clone()`. The clone method is declared in `java.lang.Object` and does a shallow copy.

29. What is the difference between == and `.equals()`?

- == - tests to see if two reference variables refer to the exact same instance of an object.
- `.equals()` - tests to see if the two objects being compared to each other are equivalent, but they need not be the exact same instance of the same object.

30. What is an enhanced for loop and what is a `forEach` loop?

- Enhanced for loop allows easier traversal of Collections (actually any arrays or `Iterable`s) - syntax: `for (Object o : collection) {...}`

31. What are 3 usages of `super` keyword?

-
    1. to refer to immediate parent class instance variable.
-
    2. `super()` is used to invoke immediate parent class constructor (also can pass params)
-

3. to invoke immediate parent class method.

OOP

32. What are the 4 pillars of OOP / Explain each?

- Abstraction - Hiding implementation details
- Polymorphism - Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class. An object can also be referenced by its supertype "parent" class, for example `ParentClass obj = new SubClass();`
- Inheritance - A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).
- Encapsulation can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class. Access to the data and code is tightly controlled by an interface.

33. What is the difference between an abstract class and an interface?

- An abstract class can have both concrete and abstract methods whereas an interface must have only abstract methods if any (unless the default keyword is used). Interface methods are implicitly public and abstract, interface variables are implicitly public, static, and final, but these do not apply in abstract classes. Neither can be instantiated

34. What are the implicit modifiers for interface variables?

- public static final

35. What is the difference between method overloading and overriding?

- Method overriding - In a subclass when one declares an identical method from the superclass, this method overrides the one in the superclass.
- Method overloading - Within the same class when one declares more than method with the same name but different signature (parameters).

36. Can you overload / override a main method? static method? a private method? a default method? a protected method?

- Main method - overload, cannot override b/c is static method.
- Static method - overload, cannot override b/c belongs to class (not inherited).
- Private method - overload, cannot override b/c doesn't get inherited.

- Default method - both.
- Protected method - both (override if inherited).

37. What are covariant return types?

- A method is allowed to return objects that are child classes of the return type. Also, when overriding a method, the return type of the new method can be a child class of the original return type

38. Difference between extends and implements?

- Extends is for classes, implements is for implementing interfaces

39. What are enumerations (enums)?

- A special Java type that defines a collection of constants

40. What are the implicit modifiers for interface variables / methods?

- methods - public abstract; variables - public static final

41. First line of constructor?

- The compiler will insert `super()` as the first line - it cannot be used anywhere else in constructor except for the first line

Collections / Generics

42. What are collections in Java?

- A general data structure that contains Objects. Also the name of the API

43. What are the interfaces in the Collections API?

- Iterable, Collection, List, Queue, Set, Map, SortedSet, SortedMap

44. What is the difference between a `Set` and a `List`?

- `Set` does not allow duplicates (its members are unique)

45. What is the difference between a `Array` and an `ArrayList`?

- An array is static and its size cannot be changed, but an ArrayList can grow/shrink

46. What is the difference between `ArrayList` and `Vector`?

- `Vector` is synchronized whereas `ArrayList` is not.

47. What is the difference between `TreeSet` and `HashSet`?

- The two general purpose `Set` implementations are `HashSet` and `TreeSet`. `HashSet` is much faster (constant time versus log time for most operations) but offers no ordering guarantees.

48. What is the difference between HashTable and HashMap?

- a. `Hashtable` is synchronized whereas `Hashmap` is not.
- b. `Hashmap` permits `null` values and the `null` key.

49. Are Maps in the Collections API?

- Yes, but they do not implement `Collection` or `Iterable` interfaces

50. What are generics? What is the diamond operator (<>)?

- A way of specifying a type within a data structure - they enforce type safety. <> operator lets you infer generic types from the LHS of assignment operation

Threads

51. What is multi-threading?

- Handling multiple threads / paths of execution in your program.

52. In what ways can you create a thread?

- By extending the Thread Class or by implementing the `Runnable` Interface. You must call Thread's `.start()` method to start it as a new thread of execution.

53. Lifecycle of a thread

- When created, in NEW state.
- When `.start()` is called, it goes to RUNNABLE state.
- When `.run()` is called, goes to RUNNING state.
- If `.sleep()` or `.wait()` is called, will go to WAITING.
- If dependent on another thread to release a lock, it will go to BLOCKED state.
- When finished executing, will be in DEAD state and cannot be restarted.

54. What is deadlock?

- When two or more threads are waiting on locks held by the others, such that no thread can execute

55. What is synchronized keyword?

- Only allowing one thread access to the method or variable at a time - enforces thread-safety

IO / Serialization

56. How do you serialize / deserialize an object in Java?

- a. Step 1: An object is marked serializable by implementing the `java.io.Serializable` interface, which signifies to the underlying API that the object can be flattened into bytes and subsequently inflated in the future.
- b. Step 2: The next step is to actually persist the object. That is done with the `java.io.ObjectOutputStream` class. That class is a filter stream--it is wrapped around a lower-level byte stream (called a node stream) to handle the serialization protocol for us. Node streams can be used to write to file systems or even across sockets. That means we could easily transfer a flattened object across a network wire and have it be rebuilt on the other side!
- c. To restore the object back, you use `ObjectInputStream.readObject()` method call. The method call reads in the raw bytes that we previously persisted and creates a live object that is an exact replica of the original. Because `readObject()` can read any serializable object, a cast to the correct type is required. With that in mind, the class file must be accessible from the system in which the restoration occurs. In other words, the object's class file and methods are not saved; only the object's state is saved.

57. What is a Marker interface?

- A marker interface is an interface which has no methods at all.
  Example: `Serializable`, `Remote`, `Cloneable`. Generally, they are used to give additional information about the behavior of a class.

58. What are transient variables?

- Transient variables are those variables which cannot be serialized.

59. Difference between FileReader and BufferedReader?

- `FileReader` is just a `Reader` which reads a file, so it reads characters and uses the platform-default encoding.

- `BufferedReader` reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines (e.g. can read one line at a time).
- So you can wrap a `BufferedReader` around a `FileReader`

Exceptions

60. What is the difference between `final`, `.finalize()`, and `finally`?

- a. `final`: final keyword can be used for class, method and variables. A final class cannot be subclassed and it prevents other programmers from subclassing a secure class to invoke insecure methods. A final method can't be overridden. A final variable can't change from its initialized value.
- b. `finalize()`: finalize method is used just before an object is destroyed and called just prior to garbage collection.
- c. `finally`: finally, a key word used in exception handling, creates a block of code that will be executed after a `try/catch` block has completed and before the code following the `try/catch` block. The `finally` block will execute whether or not an exception is thrown. For example, if a method opens a file upon exit, then you will not want the code that closes the file to be bypassed by the exception-handling mechanism. This finally keyword is designed to address this contingency.

61. `throw` vs `throws` vs `Throwable`?

- `Throwable` - the root interface of exceptions, allow a class to be "thrown"
- `throws` - keyword in method signature after params that declare which exception the method might throw
- `throw` - the keyword that will actually "throw" an exception in code

62. Do you need a catch block? Can have more than 1? Order of them?

- Catch block is not necessary - try/finally will compile. You can have more than one catch block, but the order must be from most narrow exception to most broad/general.

63. What is base class of all exceptions? What interface do they all implement?

- The base class is `Exception`, which implements the `Throwable` interface

64. List some checked and unchecked exceptions?

- Checked - `IOException`, `ClassNotFoundException`, `InterruptedException`

- Unchecked
  - `ArithmeticException`, `ClassCastException`, `IndexOutOfBoundsException`, `NullPointerException`

65. Multi-catch block - can you catch more than one exception in a single catch block?

- Yes, use the `|` operator

Reflections API

66. What is Reflection API?

- The first component of the Reflection API is the mechanism used to fetch information about a class. This mechanism is built into the class named Class. The special class Class is the universal type for the meta information that describes objects within the Java system. Class loaders in the Java system return objects of type Class. Up until now the three most interesting methods in this class were:
- `.forName()`, which would load a class of a given name, using the current class loader
- `.getName()`, which would return the name of the class as a String object,which was useful for identifying object references by their class name
- `.newInstance()`, which would invoke the null constructor on the class (if it exists) and return you an object instance of that class of object
- To these three useful methods the Reflection API adds some additional methods to class Class. These are as follows:
- `getConstructor`, `getConstructors`, `getDeclaredConstructor`
- `getMethod`, `getMethods`, `getDeclaredMethods`
- `getField`, `getFields`, `getDeclaredFields`
- `getSuperclass`
- `getInterfaces`
- `getDeclaredClasses`

Design patterns

67. What are Singleton / Factory design patterns?

- Singleton - allows for creation of only 1 object. Method for retrieving object returns reference to the same object in memory. Implement via private constructor
- Factory - abstracts away instantiation logic, usually used in conjunction with singleton pattern

JDBC

68. What is JDBC?

- A Java API used to execute queries on various databases. Uses JDBC drivers to connect with the database

69. What are the core interfaces / classes in JDBC?

- `DriverManager`, `Connection`, `Statement`, `PreparedStatement`, `CallableStatement`, `ResultsSet`

70. What is a stored procedure and how would you call it in Java?

- A stored procedure is an executable block of code that is written in PL/SQL and stored in the Oracle database. A stored procedure is called from a Java class using a CallableStatement object. When the procedure is called, its name and any relevant parameters are sent over the JDBC connection to the DBMS, which executes the procedure and returns the results (if applicable) via the connection.

71. What is the difference between Statement and PreparedStatement?

- PreparedStatements are pre-compiled by the JVM. The database doesn't have to compile the SQL each and every time it is executed. PreparedStatement can be parameterized, which can make the SQL more readable. Furthermore, PreparedStatement will properly escape reserved characters to prevent SQL injection attacks.

72. Steps to executing an SQL query using JDBC?

- 1. Register the driver using `.forName()` (or let `DriverManager` detect and load automatically from classpath)
- 2. Create the connection (`DriverManager.getConnection(url,username,password)`)
- 3. Create a statement for executing the SQL query (`Statement st = conn.createStatement()`);
- 4. Execute the SQL query (`ResultSet rs = st.executeQuery(String sql)`)
- 5. Use `ResultSet` to get values returned (`rs.getInt(1)`, etc)
- 6. Close the connection (`conn.close()`)

73. How to execute stored procedures using JDBC?

- Use the `Callable` statement interface

74. Which interface is responsible for transaction management?

- The `Connection` interface - can `commit`, `rollback`, etc

JUnit

75. What is JUnit?

- A Java unit testing framework for testing code - use it for TDD

76. What is TDD?

- Test-driven development - write unit tests before application code, then write code to make tests pass. Repeat this process until functionality is complete.

77. What are the annotations in JUnit? Order of execution?

- BeforeClass, AfterClass, Before, After, Test, Ignore

78. Give an example of a test case?

- Adding two numbers, check that the method returns the sum

Log4j

79. What is an advantage to using a logging library?

- Allows you to set logging thresholds

80. What is log4j?

- Logging library for Java

81. What are the logging levels of log4j?

- TRACE, DEBUG, INFO, WARN, ERROR, FATAL

Maven

82. What is Maven?

- A build automation and dependcy management tool for Java applications

83. What is the default Maven build lifecycle?

- process resources - copy and process the resources into destination directory
- compile - compile the source code
- process-test-resources - same for test directory
- test-compile - compile the test code
- test - run the test code
- package - combine compiled source code into a .jar or .war file
- install - install package to local repo
- deploy - copy package and install in remote repo

84. Where / when does Maven retrieve dependencies from? Where are they stored locally?

- Maven first looks to see if the dependency is in the local repo under `.m2` directory. If not, it will download the necessary .jar file(s) from the remote central Maven repository into the `.m2` directory

85. What is the POM and what is the pom.xml?

- POM stands for project object model and is the model used by Maven to understand project attributes and dependencies. The pom.xml is the xml document which lists those attributes and dependencies

Advanced

86. What are functional interfaces?

- Functional interfaces only have one method, and can be used in conjuntion with lambdas

87. What are lambdas?

- Like anonymous functions, they allow implementation of functional interfaces directly without creating a class

88. What is try-with-resources? What interface must the resource implement to use this feature?

- Try-with-resources allows for automatically closing resources in a try/catch block using `try(resource) {...}` syntax. Must implement the `AutoCloseable` interface

89. How to make numbers in your code more readable?

- Use the _ for numeric literals - must be placed between numbers

90. Which collections cannot hold null values?

- `HashTable`, `TreeSet`, `ArrayDeque`, `PriorityQueue`

91. If 2 interfaces have default methods and you implement both, what happens?

- The code will NOT compile unless you override the method. However, the code WILL compile if one interface is implemented further up in the class hierarchy than the other - in this case, the closest method implementation in the hierarchy will be called

92. If 2 interfaces have same variable names and you implement both, what happens?

- The code will compile unless you make a reference to the variable (this is an ambiguous reference). You must explicitly define the variable by using the interface name: `int a = INTERFACENAME.a;`

93. Why does `HashTable` not take `null` key?

- The hash table hashes the keys given as input, and the `null` value cannot be hashed

94. What new syntax for creating variables was introduced with Java 10?

- The `var` keyword was introduced - with type inference

95. Is there an interactive REPL tool for Java like there is for languages like Python?

- Yes, the `jshell` tool introduced in Java 9

96. What are collection factory methods?

- They allow you to directly populate collections, e.g. `Set.of(1,2,3)`