Q. Can you explain how Strings are interned in Java?

Ans. String class is designed with the Flyweight design pattern in mind. Flyweight is all about re-usability without having to create too many objects in memory. A pool of Strings is maintained by the String class. When the intern( ) method is invoked, equals(..) method is invoked to determine if the String already exist in the pool. If it does then the String from the pool is returned instead of creating a new object. If not already in the string pool, a new String object is added to the pool and a reference to this object is returned. For any two given strings s1 & s2, s1.intern( ) == s2.intern( ) only if s1.equals(s2) is true.

Two String objects are created by the code shown below. Hence s1 == s2 returns false.

|  |  |
| --- | --- |
| 1  2  3  4 | //Two new objects are created. Not interned and not recommended.  String s1 = new String("A");  String s2 = new String("A"); |

s1.intern() == s2.intern() returns true, but you have to remember to make sure that you actually do intern() all of the strings that you’re going to compare. It’s easy to forget to intern() all strings and then you can get confusingly incorrect results. Also, why unnecessarily create more objects?

Instead use string literals as shown below to intern automatically:

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "A";  String s2 = "A"; |

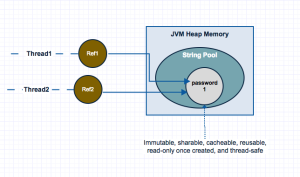
s1 and s2 point to the same String object in the pool. Hence s1 == s2 returns true.

Since interning is automatic for String literals String s1 = “A”, the intern( ) method is to be used on Strings constructed with new String(“A”).

Q. Why String class has been made immutable in Java?

A. For security, performance, and thread-safety.

1. Performance: Immutable classes are ideal for representing values of abstract data (i.e. value objects) types like numbers, enumerated types, etc. If you need a different value, create a different object. In Java, *Integer*, *Long*, *Float*, *Character*, *BigInteger* and *BigDecimal* are all immutable objects. Optimization strategies like caching of hashcode, caching of objects, object pooling, etc can be easily applied to improve performance. If String were made mutable, string pooling would not be possible as changing the string with one reference will lead to the wrong value for the other references.



String Pool

In Java 6 — all interned strings were stored in the PermGen – the fixed size part of heap mainly used for storing loaded classes and string pool.

In Java 7 – the string pool was relocated to the heap. So, you are not restricted by the limited size.

2. Thread safety as immutable classes are inherently thread safe as they cannot be modified once created. They can only be used as a read only objects. They can easily be shared among multiple threads for better scalability.

3. Errors & Security Vulnerabilities: In Java you pass sensitive information like file names, host names, login names, passwords, customer account numbers, etc as a string object. If String were not immutable, a password or account number can be accidentally & easily changed, which can cause errors and security vulnerabilities.

Q6. What can prevent execution of a code in a finally block?

A6. a) An end-less loop.

public static void main(String[ ] args) {

try {

System.out.println("This line is printed .....");

//endless loop

while(true){

//...

}

}

finally{

System.out.println("Finally block is reached."); // won't reach

}

}

public static void main(String[ ] args) {

try {

System.out.println("This line is printed .....");

//endless loop

while(true){

//...

}

}

finally{

System.out.println("Finally block is reached."); // won't reach

}

}

b) System.exit(1) statement.

public class Temp {

public static void main(String[ ] args) {

try {

System.out.println("This line is printed .....");

System.exit(1);

}

finally{

System.out.println("Finally block is reached.");// won't reach

}

}

}

public class Temp {

public static void main(String[ ] args) {

try {

System.out.println("This line is printed .....");

System.exit(1);

}

finally{

System.out.println("Finally block is reached.");// won't reach

}

}

}

c) Thread death or turning off the power to CPU.

d) An exception arising in a finally block itself.

e) Process p = Runtime.getRuntime( ).exec(““);

Q7. Can you describe “method overloading” versus “method overriding”? Does it happen at compile time or runtime?

A7. Method overloading: Overloading deals with multiple methods in the same class with the same name but different method signatures. Both the below methods have the same method names but different method signatures, which mean the methods are overloaded.

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5 | public class {  public static void evaluate(String param1); // method #1  public static void evaluate(int param1); // method #2  } |

This happens at compile-time. This is also called compile-time polymorphism because the compiler must decide how to select which method to run based on the data types of the arguments. If the compiler were to compile the statement:

|  |  |  |
| --- | --- | --- |
|  | 1  2 | evaluate(“My Test Argument passed to param1”); |

Method overriding: Overriding deals with two methods, one in the parent class and the other one in the child class and has the same name and signatures. Both the below methods have the same method names and the signatures but the method in the subclass MyClass overrides the method in the superclass BaseClass.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class A {  public int compute(int input) { //method #3  return 3 \* input;  }  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class B extends A {  @Override  public int compute(int input) { //method #4  return 4 \* input;  }  } |

This happens at runtime. This is also called runtime polymorphism because the compiler does not and cannot know which method to call. Instead, the JVM must make the determination while the code is running.

The method compute(..) in subclass “B” overrides the method compute(..) in super class “A”. If the compiler has to compile the following method,

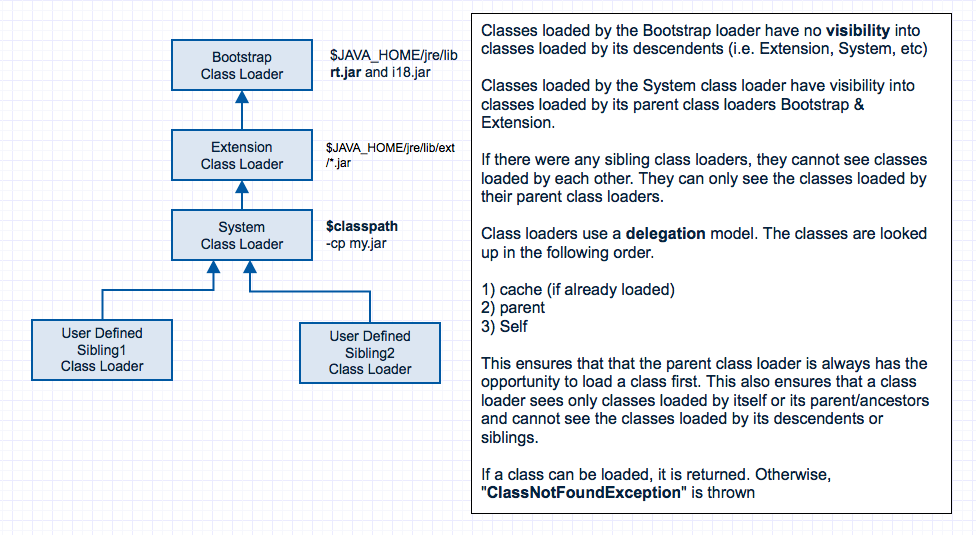
|  |  |
| --- | --- |
| 1  2  3  4 | public int evaluate(A reference, int arg2) {  int result = reference.compute(arg2);  } |

The compiler would not know whether the input argument ‘reference’ is of type “A” or type “B”. This must be determined during runtime whether to call method #3 or method #4 depending on what type of object (i.e. instance of Class A or instance of Class B) is assigned to input variable “reference”.

|  |  |
| --- | --- |
| 1  2  3  4  5 | A obj1 = new B( );  A obj2 = new A( );  evaluate(obj1, 5); // 4 \* 5 = 20. method #4 is invoked as stored object is of type B  evaluate(obj2, 5); // 3 \* 5 = 15. method #3 is invoked as stored object is of type A |

Q8. What do you know about class loading? Explain Java class loaders? If you have a class in a package, what do you need to do to run it? Explain dynamic class loading?

A8. Class loaders are hierarchical. Classes are introduced into the JVM as they are referenced by name in a class that is already running in the JVM. So, how is the very first class loaded? The very first class is specially loaded with the help of static main( ) method declared in your class. All the subsequently loaded classes are loaded by the classes, which are already loaded and running. A class loader creates a namespace. All JVMs include at least one class loader that is embedded within the JVM called the primordial (or bootstrap) class loader. The JVM has hooks in it to allow user defined class loaders to be used in place of primordial class loader. Let us look at the class loaders created by the JVM.



Class loaders are hierarchical and use a delegation model when loading a class. Class loaders request their parent to load the class first before attempting to load it themselves. When a class loader loads a class, the child class loaders in the hierarchy will never reload the class again. Hence uniqueness is maintained. Classes loaded by a child class loader have visibility into classes loaded by its parents up the hierarchy but the reverse is not true as explained in the above diagram.

Q9. Explain static vs. dynamic class loading?

A9. Classes are statically loaded with Java’s “new” operator.

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4  5  6 | class MyClass {  public static void main(String args[]) {  Car c = new Car( );  }  } |

Dynamic loading is a technique for programmatically invoking the functions of a class loader at run time. Let us look at how to load classes dynamically.

|  |  |
| --- | --- |
| 1  2  3 | //static method which returns a Class  Class.forName (String className); |

The above static method returns the class object associated with the class name. The string className can be supplied dynamically at run time. Unlike the static loading, the dynamic loading will decide whether to load the class Car or the class Jeep at runtime based on a properties file and/or other runtime conditions. Once the class is dynamically loaded the following method returns an instance of the loaded class. It’s just like creating a class object with no arguments.

|  |  |  |
| --- | --- | --- |
|  | 1  2  3  4 | // A non-static method, which creates an instance of a  // class (i.e. creates an object).  class.newInstance ( ); |

Static class loading throws “*NoClassDefFoundError*” if the class is not found and the dynamic class loading throws “*ClassNotFoundException*” if the class is not found.

Q10. What tips would you give to someone who is experiencing a class loading or “Class Not Found” exception?

A10. “*ClassNotFoundException*” could be quite tricky to troubleshoot. When you get a *ClassNotFoundException*, it means the JVM has traversed the entire classpath and not found the class you’ve attempted to reference.

1) Stand alone Java applications use -cp or -classpath to define all the folders and jar files to look for. In windows separated by “;” and in Unix separated by “:”.

|  |  |  |
| --- | --- | --- |
|  | 1  2 | java -classpath "C:/myproject/classes;C:/myproject/lib/my-utility.jar;C:/myproject/lib/my-dep.jar" MyApp |

2) Determine the jar file that should contain the class file within the classpath — war/ear archives and application server lib directories. Search recursively for the class.

|  |  |  |
| --- | --- | --- |
|  | 1  2 | $ find . -name "\*.jar" -print -exec jar -tf '{}' \; | grep -E "jar$|String\.class" |

3) Check the version of the jar in the manifest file MANIFEST.MF, access rights (e.g. read-only) of the jar file, presence of multiple versions of the same jar file and any jar corruption by trying to unjar it with “jar -xvf …”. If the class is dynamically loaded with Class.forName("com.myapp.Util"), check if you have spelled the class name correctly.

4) Check if the application is running under the right JDK? Check the JAVA\_HOME environment property

|  |  |
| --- | --- |
| 1  2 | $ echo $JAVA\_HOME |

5) -verbose:class option in your JVM. With the -verbose option all the classes that are loaded are listed, along with the JAR file or directory from which they were loaded. The “class” output shows additional information, such as when superclasses are being loaded, and when static initializers are being run.

6) Creating a Java dump and analyzing the Java dump for class loading issues. The Java dumps are created under following circumstances.

— When a fatal native JVM error is thrown.

— When the JVM runs out of heap memory space.

— When a signal is sent to the JVM (e.g. Control-Break is pressed on Windows, Control-\ on Linux, or kill -3 on Unix)

There are tools like jstack, jmap, hprof, and Eclipse Memory Analyzer (MAT) to analyze the Java dumps.

7) Some of the libraries provide API to list the version number. For example, The Eclipse link MOXy library provides a method as shown below.

|  |  |
| --- | --- |
| 1 | org.eclipse.persistence.Version.getVersion(); |

8) The org.jboss.test.util.Debug class has a method *displayClassInfo(Class clazz, StringBuffer results)* to display the loaded class details. This is done programmatically. What this class essentially does is

URL loc = MyClass.class.getProtectionDomain().getCodeSource().getLocation();

9) The http://www.findjar.com is an online search engine that can list possible jar files in which a particular class file like java.sql.Connection can be found.

**Q. FizzBuzz problem :** Write a Java program that prints the numbers from 1 to 50. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"

This is also one of the classical programming questions, which is asked on any Java programming or technical interviews. This questions is very basic but can be very trick for programmers, who can't code, that's why it is used to differentiate programmers who can do coding and who can't. Here is a sample Java program to solve FizzBuzz problem :

Sol:

public class FizzBuzzTest {

public static void main(String aa[]) {

for(int i=1; i<=50; i++){

if(i%(3\*5)== 0)

System.out.println("FizzBuzz");

else if(i%5 == 0)

System.out.println("Buzz");

else if(i%3 == 0)

System.out.println("Fizz");

else

System.out.println(i);

}

}

}

Q. Write a Comparator in Java to compare two employees based upon there name, departments and age?

Sol:

Q. Java program to find even and odd number in Java

Sol:

package com;

import java.util.Scanner;

public class EvenOddTest {

public static void main(String aa[]) {

Scanner console = new Scanner(System.in);

System.out.println("Enter Any Number : ");

int number = console.nextInt();

if (number % 2 == 0)

System.out.printf("Number %d is even number %n", number);

else

System.out.printf("Number %d is odd number %n", number);

System.out.printf("Finding number if its even or odd using bitwise AND operator %n");

if ((number & 1) == 0)

System.out.printf("Number %d is even number %n", number);

else

System.out.printf("Number %d is odd number %n", number);

}

}

**Q. Write a program in Java to find out if a number is prime in Java?**

**Sol:**

**Q. Write Java program to check if a number is palindrome in Java?**

**Sol:**

package com;

import java.util.Scanner;

public class NoClassDefFoundErrorDueToStaticInitFailure {

public static void main(String args[]){

System.out.println("Please Enter a number : ");

int palindrome = new Scanner(System.in).nextInt();

if(isPalindrome(palindrome)){

System.out.println("Number : " + palindrome + " is a palindrome");

}else{

System.out.println("Number : " + palindrome + " is not a palindrome");

}

}

public static boolean isPalindrome(int number) {

int palindrome = number;

int reverse = 0;

while (palindrome != 0) {

int remainder = palindrome % 10;

reverse = reverse \* 10 + remainder;

palindrome = palindrome / 10;

}

if (number == reverse) {

return true;

}

return false;

}

}

Q. Write program to sort an integer array without using API methods?

Sol:

Bubble sort in Java - program to sort integer array

Bubble sort is one of the classic sorting algorithm which is used to explain sorting during various computer and engineering courses. Because of its algorithmic nature and simplicity its often used in various Java and C++ programming exercises. You may expect questions like Write Java program to sort integer array using bubble sort during any programming interview. Since algorithmic questions are always tricky question and not easy to code. Even simplest of them can lead to confusion, especially if you are not gifted with a natural programming head. I have seen many developers fumble if asked to code on the spot. That's why its advisable to do algorithmic and logical programming during training and learning programming and OOPS to get this skill of converting logic into code. Let's come back to Bubble sort, In Bubble sort algorithm we sort an unsorted array by starting from first element and comparing with adjacent element. If former is greater than later then we swap and by doing this we get the largest number at the end after first iteration. So in order to sort n elements you require n-1 iteration and almost n-1 comparison. For recap here is the logic for bubble sort sorting algorithm :

1) start comparing a[0] to a[1]

2) if a[0] > a[1] then swap numbers e.g. a[0]=a[1] and a[1]=a[0]

3) compare a[1] to a[2] and repeat till you compare last pair

4) This is referred as one pass and at the end of first pass largest number is at last

5) repeat this comparison again starting from a[0] but this time going till second last pair only

Now let's see Java program which implements this bubble sort logic to sort unsorted integer array.

package com;

import java.util.Arrays;

public class BubbleSort {

public static void main(String args[]) {

//testing our bubble sort method in Java

int[] unsorted = {32, 39,21, 45, 23, 3};

bubbleSort(unsorted);

//one more testing of our bubble sort code logic in Java

int[] test = { 5, 3, 2, 1};

bubbleSort(test);

}

/\*

\* In bubble sort we need n-1 iteration to sort n elements

\* at end of first iteration larget number is sorted and subsequently numbers smaller

\* than that.

\*/

public static void bubbleSort(int[] unsorted){

System.out.println("unsorted array before sorting : " + Arrays.toString(unsorted));

// Outer loop - need n-1 iteration to sort n elements

for(int i=0; i<unsorted.length -1; i++){

//Inner loop to perform comparision and swapping between adjacent numbers

//After each iteration one index from last is sorted

for(int j= 1; j<unsorted.length -i; j++){

//If current number is greater than swap those two

if(unsorted[j-1] > unsorted[j]){

int temp = unsorted[j];

unsorted[j] = unsorted[j-1];

unsorted[j-1] = temp;

}

}

System.out.printf("unsorted array after %d pass %s: %n", i+1, Arrays.toString(unsorted));

}

}

}

**6. Write Java program to check if a number is Armstrong number or not?**

Sol:

package com;

import java.util.Scanner;

public class ArmstrongTest {

public static void main(String args[]) {

// input number to check if its Armstrong number

System.out.println("Please enter a 3 digit number to find if its an Armstrong number:");

int number = new Scanner(System.in).nextInt(); // printing result

if (isArmStrong(number)) {

System.out.println("Number : " + number + " is an Armstrong number");

} else {

System.out.println("Number : " + number + " is not an Armstrong number");

}

}

private static boolean isArmStrong(int number) {

int result = 0;

int orig = number;

while (number != 0) {

int remainder = number % 10;

result = result + remainder \* remainder \* remainder;

number = number / 10;

} // number is Armstrong return true

if (orig == result) {

return true;

}

return false;

}

}

7. Write a program in Java to reverse any String without using StringBuffer?

Sol:

package com;

public class StringReverseExample {

public static void main(String args[]) {

// quick wasy to reverse String in Java - Use StringBuffer

String word = "HelloWorld";

String reverse = new StringBuffer(word).reverse().toString();

System.out.printf(" original String : %s , reversed String %s %n", word, reverse);

// another quick to reverse String in Java - use StringBuilder

word = "WakeUp";

reverse = new StringBuilder(word).reverse().toString();

System.out.printf(" original String : %s , reversed String %s %n", word, reverse);

// one way to reverse String without using StringBuffer or StringBuilder

// is writing

// own utility method

word = "Band";

reverse = reverse(word);

System.out.printf(" original String : %s , reversed String %s %n", word, reverse);

}

public static String reverse(String source) {

if (source == null || source.isEmpty()) {

return source;

}

String reverse = "";

for (int i = source.length() - 1; i >= 0; i--) {

reverse = reverse + source.charAt(i);

}

return reverse;

}

}

8. Write a program in Java to print Fibonacci series up to given number? Write both iterative and recursive version.

Sol: