|  |  |
| --- | --- |
|  | **Lesson 1:Introduction to Java** |
| **Q. 1** | **Define :**   1. **Bytecode verifier** 2. **Class loader** 3. **JIT compiler** 4. **JVM** 5. **JDK** |
| Ans : | 1. Component of the JVM that ensures that byte code does not access private data 2. Responsible for keeping classes from different servers separate from each other as well as the local classes |
|  |  |
|  | **Lesson 3: Language Fundamentals** |
| **Q. 2** | **What is the default value for float , char ,byte data types ?** |
| **Ans :** | int = 0  String = null  float = 0.0  double = 0.0  boolean=false |
|  |  |
| Q. 3 | a) What happens when you ignore break in switch. Given an Example.  b) Discuss the valid data types that w.r.t switch case ? |
| **Ans :** | 1. All the following cases will get executed till it finds the break statement.   **Example :** |
|  | **Lesson 4: Classes and Objects** |
| **Q 4** | What is the default value of all instance variables |
| **Ans.** |  |
| **Q5** | What will happen if you don’t initialize a local variable and try to print it? |
| **Ans.** |  |
| **Q6** | What is the need of :   1. Final 2. Finalize 3. Finally |
| **Ans.** |  |
| **Q7** | Discuss Static method. |
|  |  |
|  | What is created in heap |
| **Ans** | Object |
|  |  |
|  | **Lesson 5: Exploring Basic Java Class Libraries** |
| **Q.8** | What are wrapper classes, list all the wrapper classes |
| **Ans** |  |
| **Q9** | What are the possible modifiers for :   1. class/static variables 2. instance variables 3. local variables   (example : final is the only modifier used for local variables and public, private and static is used for local) |
| **Ans** |  |
| **Q.10** | Which type of variables must be initialized-mandatory ? |
| **Ans** |  |
| **Q.11** | Which of the following are mutable/immutable ? |
|  | 1. string 2. String buffer 3. String builder |
| **Ans:** |  |
| **Q.12** | Give examples of using :   1. Append 2. Concat 3. Equals 4. == 5. compareTo |
| **Ans** |  |
| **Q13** | 1. Give example of using Scanner object. 2. State all the nextxxx() methods used with scanner object |
| **Ans** |  |
| **Q14** | Give examples for each :   1. Get current date 2. Get tomorrows date - Add one day 3. Get yesterdays date -Subtract one day |
| **Ans** |  |
| **Q. 15** | State and give Examples for all Object class methods. |
| **Ans** |  |
|  | **Lesson 6: Inheritance and Polymorphism** |
| **Q.16** | Difference between overriding and overloading |
| **Ans** |  |
| **Q.17** | **Difference between Abstract Class and Interface** |
| **Ans** |  |
| **Q. 18** | **State the modifiers of the data members in a interface** |
| **Ans** |  |
| **Q.19** | Aggregation relationship – how will you implement in java |
|  |  |
| **Q.20** | InstanceOf – Use and Example |
| **Ans** |  |
| **Q.21** | Discuss all points about key word “this” and “super” with examples (while writing constructors) |
| **Ans** |  |
| **Q.22** | How will you write varargs (what conditions must be followed). |
| **Ans** |  |
| **Q.23** | Explain all points about :   1. Final variable 2. Final method 3. Final class |
| **Ans** |  |
|  | **Lesson 7: Abstract Classes and Interfaces** |
| **Q24** | Difference between final and abstract class |
| **Ans** |  |
| **Q.25** | By default interface data members are \_\_\_\_\_\_\_\_\_\_\_ |
| **Ans** |  |
|  | **Lesson 8:Regular Expressions** |
| **Q.26** | Three classes in regex package |
| **Ans.** |  |
| **Q.27** | Examples on pattern matching |
| **Ans.** |  |
|  | **Lesson 9:Exception Handling** |
| **Q.28** | List and discuss ALL the Checked exception and UnChecked exception |
| **Ans.** | * Checked Exceptions : SQLException, IOException, ClassNotFoundException * UnChecked Exceptions : NullPointerException, ArithmeticException, ArrayIndexOutOfBoundException, NumberFormatException |
| **Q.29** | Base class of all exception |
| **Ans.** | Throwable |
| **Q.30** | How will you create checked and unchecked user defined exception |
| **Ans.** | The Java compiler enforces a catch-or-declare requirement for checked exceptions. An exception's type determines whether the exception is checked or unchecked |
| **Q.31** | Define :   1. Try 2. Catch 3. Finally 4. Throw 5. Throws |
| **Ans.** |  |
| **Q.32** | Significance of Try-with-resource feature in exception handling |
| **Ans.** | A try-with-resources is a new feature added in java 7, where resources are closed automatically. Any block after try (either catch or finally block) will be executed only after the resource is closed |
| **Q.33** | Any null reference with method invocation will create NullPointer exception – Give example |
| **Ans.** | class TryCatchDemo{  public static void main(String a[]) {  String str= null;  try {  str.equals("Hello");  } catch(NullPointerExceptionne) {  str= new String("Hello");  System.out.println(str.equals("Hello"));  }  System.out.println("Continuing in the program");  }  } |
| **Q.34** | Layered architecture of Exception handling |
| **Ans.** |  |
|  | **Lesson 10:Array** |
| **Q.35** | Syntax for declaring and initializing arrays of various fundamental data types. Give example |
| **Ans.** | **One-Dimensional Arrays :.**  type var-name[];  OR  type[] var-name;  eg:  int intArray[]; //declaring array  intArray = new int[20]; // allocating memory to array  **Multidimensional Arrays**   A multidimensional array is created by appending one set of square brackets ([]) per dimension.  Eg:  int[][] intArray = new int[10][20]; //a 2D array or matrix  int[][][] intArray = new int[10][20][10]; //a 3D array  Using varargs  public static void fun(int ... a)  {  // method body  }  Eg:  // Java program to demonstrate varargs  class Test1  {      // A method that takes variable number of intger      // arguments.      static void fun(int ...a)      {          System.out.println("Number of arguments: " + a.length);            // using for each loop to display contents of a          for (int i: a)              System.out.print(i + " ");          System.out.println();      }        // Driver code      public static void main(String args[])      {          // Calling the varargs method with different number          // of parameters          fun(100);         // one parameter          fun(1, 2, 3, 4);  // four parameters          fun();            // no parameter      }  }  **Using Arrays class:**  **public static String toString(int[] a)**  **Eg:**  import java.util.Arrays;    public class Main  {      public static void main(String[] args)      {          int ar[] = {4, 6, 1, 8, 3, 9, 7, 4, 2};            // To print the elements in one line          System.out.println(Arrays.toString(ar));      }  } |
|  | **Lesson 11: Collection** |
| **Q.36** | Difference between enhanced for loop and iterator with example. |
| **Ans.** | [**Iterator**](http://geeksquiz.com/how-to-use-iterator-in-java/) is an interface provided by collection framework to traverse a collection and for a sequential access of items in the collection. While enhanced [**For**](https://www.geeksforgeeks.org/for-each-loop-in-java/) loop is meant for traversing items in a collection.  Eg with iterator  // Java program to demonstrate working of nested iterators  // may not work as expected and throw exception.  import java.util.\*;    public class Main  {      public static void main(String args[])      {          // Create a link list which stores integer elements          List<Integer> l = new LinkedList<Integer>();            // Now add elements to the Link List          l.add(2);          l.add(3);          l.add(4);            // Make another Link List which stores integer elements          List<Integer> s=new LinkedList<Integer>();          s.add(7);          s.add(8);          s.add(9);            // Iterator to iterate over a Link List          for (Iterator<Integer> itr1=l.iterator(); itr1.hasNext(); )          {              for (Iterator<Integer> itr2=s.iterator(); itr2.hasNext(); )              {                  if (itr1.next() < itr2.next())                  {                      System.out.println(itr1.next());                  }              }          }      }  }  Enhanced for loops   |  | | --- | | // Java program to demonstrate working of nested for-each  import java.util.\*;  public class Main  {      public static void main(String args[])      {          // Create a link list which stores integer elements          List<Integer> l=new LinkedList<Integer>();            // Now add elements to the Link List          l.add(2);          l.add(3);          l.add(4);            // Make another Link List which stores integer elements          List<Integer> s=new LinkedList<Integer>();          s.add(2);          s.add(4);          s.add(5);          s.add(6);            // Iterator to iterate over a Link List          for (int a:l)          {              for (int b:s)              {                  if (a<b)                      System.out.print(a + " ");              }          }      }  } | |
| **Q.37** | All collections’ comparison for ordered, sorted, duplicates, synchronization, key/value pair, allows null |
| **Ans.** |  |
| **Q.38** | Give an Example on :   1. Collections.sort(),Arrays.sort(array) 2. Clear() 3. removeAll() 4. isEmpty() |
| **Ans.** | Collections.sort():-  import java.util.Collections;  import java.util.Vector;  public class SortingVectorExample {  public static void main(String[] args) {  // Create a Vector  Vector<String> vector = new Vector<String>();  //Add elements to Vector  vector.add("Walter");  vector.add("Anna");  vector.add("Hank");  vector.add("Flynn");  vector.add("Tom");    // By Default Vector maintains the insertion order  System.out.println("Vector elements before sorting: ");  for(int i=0; i < vector.size(); i++){  //get(i) method fetches the element from index i  System.out.println(vector.get(i));  }  // Collection.sort() sorts the collection in ascending order  Collections.sort(vector);    //Display Vector elements after sorting using Collection.sort  System.out.println("Vector elements after sorting: :");  for(int i=0; i < vector.size(); i++){  System.out.println(vector.get(i));  }  }  }  **Output:**  Vector elements before sorting:  Walter  Anna  Hank  Flynn  Tom  Vector elements after sorting: :  Anna  Flynn  Hank  Tom  Walter  **Arrays.sort():-**  import java.util.Arrays;    public class SortExample  {      public static void main(String[] args)      {          // Our arr contains 8 elements          int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};  Arrays.sort(arr);  System.out.printf("Modified arr[] : %s",                            Arrays.toString(arr));      }  } |
|  | **Lesson 12: File IO** |
| **Q.39** | Different types of streams in File IO, Buffered Streams. |
| **Ans.** |  |
| **Q.40** | Need of flush() & isFile() method with Examples. |
| **Ans.** |  |
| **Q.41** | Difference between Serialization and Deserialization |
| **Ans.** |  |
| **Q.42** | Which classes are available in the java.io package ? |
| **Ans.** |  |
|  | **Lesson 13: Introduction to Junit 4 & Lesson 14: Advanced Testing** |
| **Q.43** | Explain @Test with all attributes like timeout, expected. |
| **Ans.** |  |
| **Q.44** | Explain @ignore |
| **Ans.** |  |
| **Q.45** | Explain static import of Assert class |
| **Ans.** |  |
| **Q.46** | Explain : '@RunWith(Suite.class), @Suite.SuiteClasses |
| **Ans.** | Define :  @Before  @After  @BeforeClass  @AfterClass |
|  |  |
| **Q.47** | What is parameterized test? |
| **Ans.** |  |