JAVA ASSIGNMENT SOLUTIONS

ANSWERS FOR MCQS:

1. A. Bytecode is executed by JVM

2. C. Use of Pointers

3. A. interface

4. A. 0.0d

5. B. class variable, local variable, instance variable

6. C. null

7. C. 32 bit

8. A. Static Binding

9. B. RunTimeError:NoSuchMethodError

10. B. Serialization is the process of writing the state of an object to a byte stream

11. A. A class is a blue print from which individual objects are created. A class can contain fields and methods to describe the behaviour of an object.

12. C. Immutable variable

13. C. It refers to the ability to make a class abstraction in OOP.

14. C. char c3=\u0022; is not a valid declaration and remaining all are valid declarations

15. B. concat()

16. C.static void methoda(double d1); is the valid declaration within an interface definition.

17. C. if(i==5)

18. C. program

19. B. James Gosling

20. C. Oak

**Q1. Write a program to scan two numbers and then perform arithmetic operations ?**

Program: Arithmetic operations

**package** javaassignmentquestions;

**import** java.util.\*;

**public** **class** arithmetic1 {

**public** **static** **void** main(String[] args)

{

**int** a,b;

**float** d;

**char** c;

Scanner s =**new** Scanner(System.***in***);

System.***out***.print("enter a and b values");

a=s.nextInt();

b=s.nextInt();

System.***out***.print("enter operator");

c=s.next().charAt(0);

**switch**(c)

{

**case** '+': d=a+b;

System.***out***.println("Addition of"+a+"and"+b+"is"+(**int**)d);

**break**;

**case** '-': d=a-b;

System.***out***.println("Subtraction of"+a+"and"+b+"is"+(**int**)d);

**break**;

**case** '\*' :d=a\*b;

System.***out***.println("Multiplication of"+a+"and"+b+"is"+(**int**)d);

**break**;

**case** '/':d=a/b;

System.***out***.println("Division of+"+a+"and"+b+"is"+(**int**)d);

**break**;

**case** '%':d=a%b;

System.***out***.println("Modulo division of "+a+"and"+b+"is"+(**int**)d);

**break**;

**default**: System.***out***.println("You entered wrong operator");

}

}

}

Output:

enter a and b values 3 4

enter operator\*

Multiplication of3and4is12

**Q2. How is Hashmap implemented in java ?**

1. **The HashMap Class:**

The **HashMap** class extends **AbstractMap** and implements the **Map** interface. It uses a hash table to store the map. This allows the execution time of **get()** and **put()** to remain constant even for large sets. **HashMap** is a generic class that has this declaration:

Class HashMap<K, V>

Here, **K** specifies the type of keys, and **V** specifies the type of values.

The following constructors are defined:

HashMap()

HashMap(Map<? Extends **K,** ? extends **V**> m)

HashMap(int *capacity*)

HashMap(int *capacity*, float *fillRatio*)

The first form constructs a default hash map. The second form initializes the hash map by using the elements of m. The third form initializes the capacity of the hash map to *capacity*. The fourth form initializes both the capacity and fill Ratio of the hash map by using its arguments. The meaning of capacity and fill Ratio is the same as for **HashSet,** described earlier. The default capacity is 16. The default fill ratio is 0.75.

**HashMap** implements **Map** and extends **AbstractMap**. It does not add any methods of its own.

You should note that a hash map does not guarantee the order if its elements.

Therefore, the order in which elements are added to a hash map is not necessarily the order in which they are read by an iterator.

The following program illustrates **HashMap**. It maps names to account balances. Notice how a set-view is obtained and used.

Program: HashMap implementation.

import java. util.\*;

class HashMapDemo{

public static void main(String args[])

{

// Create a hash map.

HashMap<String, Double> hm=new HashMap<String, Double>();

//Put elements to the map

hm.put(“Jessie”, new Double(32323.32));

hm.put(“Angel”, new Double(18929.23));

hm.put(“Bertha”, new Double(34365.32));

hm.put(“Caroline”, new Double(34525.36));

hm.put((“Diana”, new Double(-19.08));

//Get a set of the entries.

Set<Map.Entry<String, Double>> set=hm.entrySet();

//Display the set.

For(Map.Entry<String, Double> me: set){

System.out.print (me.getKey() + “: “);

System. out. println (me.getValue());

}

System.out.println();

//Deposit 1000 into Jessie’s account.

double balance= hm.get(“Jessie”);

hm.put(“Jessie”, balance+1000);

System.out.println(“Jessie’s new balance:”+hm.get(“Jessie”));

}

}

Output from this program is shown here(the precise order may vary):

Diana: -19.08

Bertha: 34365.32

Jessie: 32323.32

Caroline: 34525.36

Angel: 18929.23

Jessie’s new balance: 33323.32

The program begins by creating a hash map and then adds the mapping of names to balances. Next, the contents of the map are displayed by using a set-view, obtained by calling entrySet().

The keys and values are displayed by calling the getKey() and getValue() methods that are defined by Map.Entry. Pay close attention to those how the deposit is made into Jessie’s account. The put() method automatically replaces any preexisting value that is associated with the specified key with the new value. Hus, after Jessie’s account is updated, the hash map will still contain just one “Jessie” account.

**Q3. How do you find if an ArrayList contains an object or not ? Practical**

**Program**: Finding ArrayList contains an object or not.

**package** javaassignmentquestions;

**import** java.util.\*;

**public** **class** ArrayListobject {

**public** **static** **void** main(String args[])

{

ArrayList<Object> al=**new** ArrayList<Object>();

Object o=**new** Object();

al.add(o);

**if**(al.contains(o))

{

System.***out***.println("the ArrayList contains object");

}

}

}

**Output**:

The ArrayList contains object

**Q4. How do you sort a collection in java? practical**

Program:Sorting a collection in java

**package** javaassignmentquestions;

**import** java.util.\*;

**public** **class** collectionsorting {

**public** **static** **void** main(String[] args)

{

ArrayList<String> al= **new** ArrayList<String>();

al.add("Ball");

al.add("Cat");

al.add("Apple");

al.add("Dog");

al.add("Elephant");

Collections.*sort*(al);

System.***out***.println("the collections in ArrayList after sorting: "+al);

}

}

Output:

the collections in ArrayList after sorting: [Apple, Ball, Cat, Dog, Elephant]

**Q5.How do you remove an object from collection? Practical**

Program: Removing object from collection

**package** javaassignmentquestions;

**import** java.util.ArrayList;

**public** **class** removingobject {

**public** **static** **void** main(String[] args)

{

ArrayList<Object> al=**new** ArrayList<Object>();

Object o1= **new** Object();

Object o2=**new** Object();

Object o3=**new** Object();

al.add(o1);

al.add(o2);

al.add(o3);

al.remove(o1);

System.***out***.println("the objects in collection are:"+al);

}

}

output:

the objects in collection are:[java.lang.Object@2c7b84de, [java.lang.Object@3fee733d](mailto:java.lang.Object@3fee733d)]

**Q1. What are the main differences between array and collection?**

1. Differences between array and collection are:

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Arrays** | **Collection Framework** |
| 1 | Arrays are fixed in size. | Collections are growable in nature. |
| 2 | Arrays can contains both primitives and object type. | Collections can hold only objects but not primitive. |
| 3 | Arrays are better than collections in performance. | Collections are not poor in performance than Arrays. |
| 4 | Arrays can hold only homogeneous data types elements. | Collection can hold both homogeneous and heterogeneous elements. |
| 5 | If we know the size of elements then it is good to go for arrays. | If we are not sure about size we will go for collections. |
| 6 | Underlying data structure is not available. | Collections are implements on a standard data structure. |
| 7 | **Example**:int[] arr=new int[5]; | **Example**: ArrayList<String> alist= new ArrayList<String>(); |

Q2.Explain various interfaces used in collection framework?

1. The interfaces defined by Java.util are shown next:

|  |  |  |
| --- | --- | --- |
| *Collection* | *Map.Entry* | *Set* |
| *Comparator* | *NavigableMap* | *SortedMap* |
| *Deque* | *NavigableSet* | *SortedSet* |
| *Enumeration* | *Observer* | *Spliterator(Added by JDK 8)* |
| *EventListener* | *primitiveIterator(Added by JDK8)* | *Spliterator.OfDouble(Added by JDK 8)* |
| *Formattable* | *PrimitiveIterator.OfDouble(Added by JDK 8)* | *Spliterator.OfInt(Added by JDK 8)* |
| *Iterator* | *PrimitiveIterator.OfInt(Added by JDK8)* | *Spliterator.OfLong(Added by JDK 8)* |
| *List* | *PrimitiveIterator.OfLong(Added by JDK 8)* | *Spliterator.OfPrimitive(Added by JDK 8)* |
| *ListIterator* | *Queue* |  |
| *Map* | *RandomAccess* |  |

**Q3.What is the difference between ArrayList, Vector, ArrayList, and LinkedList?**

1. Comparison between ArrayList and Vector

|  |  |  |
| --- | --- | --- |
| **Parameter of Comparison** | **ArrayList** | **Vector** |
| Synchronization | ArrayList us not synchronized i.e., it could work on various threads simultaneously. | Vector is synchronized i.e., only one thread could handle the code at a moment. |
| Speed | Its operations are fast as they are non-synchronized. | Vector operations run slower as they are synchronized. |
| Resizing | If elements exceed their capacity then ArrayList increases 50% of the existing array size. | If elements exceed their capacity then the vector increases 100% of the existing array size. |
| Preference | Programmers prefer ArrayList over vectors. | It is less preferred as synchronization in vectors causes inferior performance. |
| Traversal(Pass-through) | It uses iterator to traverse the elements. | It can use the iterator as well as enumeration to traverse the elements. |

Differences between ArrayList and LinkedList:

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Key | ArrayList | LinkedList |
| 1 | Internal Implementation | ArrayList internally uses a  dynamic array to store its elements. | LinkedList uses Doubly Linked List to store its elements. |
| 2 | Manipulation | ArrayList is slow as array manipulation is slower. | LinkedList is faster being node based as not much bit shifting required |
| 3 | Implementation | ArrayList implements only List. | LinkedList implements List as well as Queue. It can acts as a queue as well. |
| 4 | Access | ArrayList is faster in storing and accessing data. | Linked List is faster in manipulation of data. |

**Q4. What is the difference between List and Set?**

1. Difference between List and Set:

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| 1 | The list is an ordered sequence. | The set is an unordered sequence. |
| 2 | List allows duplicate elements. | Set doesn’t allow duplicate elements |
| 3 | Elements by their position can be accessed. | Position access to elements is not allowed. |
| 4 | Multiple null elements can be stored. | Null element can store only once. |
| 5 | List implementations are ArrayList, LinkedList, Vector, Stack. | Set implementations are HashSet, LinkedHashSet. |

**Q5.Describe the newly added features of Java 8?**

A.Java 8 provides following features for Java Programming:

* Lambda expressions,
* Method references,
* Functional interfaces,
* Stream API,
* Default methods,
* Base64 Encode Decode,
* Static methods in interface,
* Optional class,
* Collectors class,
* ForEach() method,
* Nashorn JavaScript Engine,
* Parallel Array Sorting,
* Type and Repating Annotations,
* IO Enhancements,
* Concurrency Enhancements,
* JDBC Enhancements etc.

**Q6. What is lambda expression? please explain by example.**

1. **Lambda Expression:**

Lambda Expressions were added in Java 8.

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

**Syntax:**

Parameter->expression

(parameter1, parameter2)->expression

(parameter1, parameter2)->{code block}

Program:Lambda expression

**package** javaassignmentquestions;**public** **class** exampleoflambd {

**public** **static** **void** main(String[] args) {

ArrayList<Integer> numbers = **new** ArrayList<Integer>();

numbers.add(5);

numbers.add(9);

numbers.add(8);

numbers.add(1);

numbers.forEach( (n) -> { System.***out***.println(n); } );

}

}

output:

5

9

8

1

**Q7. What are functional or SAM interfaces?**

1. Java Functional Interfaces

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

Program: SAM Example

@FunctionalInterface

interface sayable{

void say(String msg);

}

Public class FunctionalInterfaceExample implements sayable{

Public void(String msg){

System.out.println(msg);

}

Public static void main(String[] args){

FunctionalInterfaceExample fie= new FunctionalInterfaceExample();

Fie.say(“Hello there”);

}

}

Output: Hello there

**Q8. How do you read from a file using FileReader class?**

We can use Reader such as FileReader for reading a text file in Java. FileReader is a convenience class meant for reading streams of characters from files that work only with the default character encoding.

To read the contents of a text file, the idea is to create a new FileReader and read the whole file character by character using Reader’s read() method, as shown below. It returns the character read, or -1 if the end of the stream is reached, and throws an IOException if an I/O error occurs.

**Program:**

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

class Main

{

public static void main(String[] args)

{

File file = new File("doc.txt");

try (FileReader fr = new FileReader(file))

{

int content;

while ((content = fr.read()) != -1) {

System.out.print((char) content);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

Output:

The text data available in the doc.txt file

**Q9. What is the use of BufferedWriter and BufferedReader classes in java?**

1. **Java BufferedWriter Class:**

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.

**Program: BufferedWriter**

**package** bw;

**import** java.io.\*;

**public** **class** BufferedWriterExample {

**public** **static** **void** main(String[] args) **throws** Exception {

FileWriter writer = **new** FileWriter("D:\\testout.txt");

BufferedWriter buffer = **new** BufferedWriter(writer);

buffer.write("Welcome to bufferedwriter example.");

buffer.close();

System.***out***.println("Success");

} }

Output:

Success

Testout.txt output:

Welcome to bufferedwriter example.

**Java BufferedReader Class:**

Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

**Program: BufferedReader example**

**package** br;

**import** java.io.\*;

**public** **class** BufferedReaderExample {

**public** **static** **void** main(String args[])**throws** Exception{

FileReader fr=**new** FileReader("D:\\testout.txt");

BufferedReader br=**new** BufferedReader(fr);

**int** i;

**while**((i=br.read())!=-1){

System.***out***.print((**char**)i);

}

br.close();

fr.close();

}

}

output of testout.txt

welcome to bufferedreader example

output:

welcome to bufferedreader example

**Q10. What is a stream and what are the types of streams and classes of the streams?**

A. **Stream:**

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

**Types of streams in java:**

Java provides I/O Streams to read and write data where, a Stream represents an input source or an output destination which could be a file, i/o devise, other program etc.

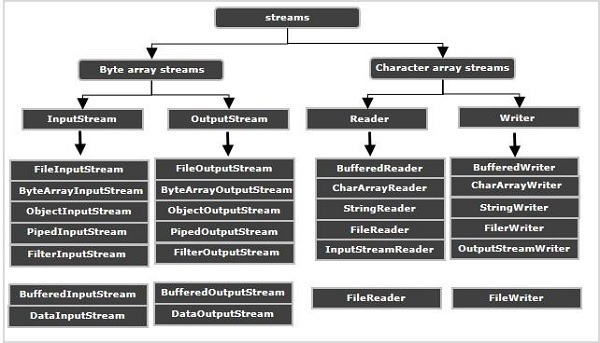
In general, a Stream will be an input stream or, an output stream.

* **InputStream** − This is used to read data from a source.
* **OutputStream** − This is used to write data to a destination.

Based on the data they handle there are two types of streams −

* **Byte Streams** − These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
* **Character Streams** − These handle data in 16 bit Unicode. Using these you can read and write text data only.

Following diagram illustrates all the input and output Streams (classes) in Java.



**Standard Streams**

In addition to above mentioned classes Java provides 3 standard streams representing the input and, output devices.

* **Standard Input** − This is used to read data from user through input devices. keyboard is used as standard input stream and represented as System.in.
* **Standard Output** − This is used to project data (results) to the user through output devices. A computer screen is used for standard output stream and represented as System.out.
* **Standard Error** − This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as System.err.

**Example**

Following Java program reads the data from user using BufferedInputStream and writes it into a file using BufferedOutputStream.

**import** java.io.BufferedInputStream;

**import** java.io.BufferedOutputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**public** **class** BufferedInputStreamExample {

**public** **static** **void** main(String args[]) **throws** IOException {

//Creating an BufferedInputStream object

BufferedInputStream inputStream = **new** BufferedInputStream(System.***in***);

**byte** bytes[] = **new** **byte**[1024];

System.***out***.println("Enter your data ");

//Reading data from key-board

inputStream.read(bytes);

//Creating BufferedOutputStream object

FileOutputStream out= **new** FileOutputStream("D:/myFile.txt");

BufferedOutputStream outputStream = **new** BufferedOutputStream(out);

//Writing data to the file

outputStream.write(bytes);

outputStream.flush();

System.***out***.println("Data successfully written in the specified file");

}

}

Output

Enter your data

Hi welcome to Streams ....

Data successfully written in the specified file

**Q1. Which of these class contains the methods used to write in a file?**

* 1. FileStream
  2. FileInputStream
  3. BUfferedOutputStream
  4. FileBufferStream

**Ans. b.FileInputStream**

**Q2.Which of these exception is thrown in cases when the file specified for writing is not found?**

1. IOException  
   b) FileException  
   c) FileNotFoundException  
   d) FileInputException

**Ans: c.FileNotFoundException**  
Explanation: In cases when the file specified is not found, then FileNotFoundException is thrown by java run-time system, earlier versions of java used to throw IOException but after Java 2.0 they throw FileNotFoundException.

**Q3.Which of these methods are used to read in from file?**a) get()b) read()  
c) scan()  
d) readFileInput()

**Ans:** b. read()  
Explanation: Each time read() is called, it reads a single byte from the file and returns the byte as an integer value. read() returns -1 when the end of the file is encountered.