Q1. What is JVM and explain the Java memory allocation?

Ans. JVM (Java Virtual Machine) is a virtual machine that runs Java bytecodes. JVM does not understand the Java source code which is why we compile the .java file to get a .class file that contains Java bytecodes understandable by JVM which also makes it portable.

Java bytecode is the instruction set of JVM. Each byte code consists of one or two bytes that represent the instruction (opcode).

***Memory Allocation***

In Java, all objects are dynamically allocated to a heap. When we only declare a variable of class type it only creates a reference, the memory is only allocated once we use new ().

Q2. What is polymorphism and encapsulation?

Ans.  ***Encapsulation*** is a process to wrap the code and data together into a single unit. We can make the class fully encapsulated by making its members private and using getter and setter methods to access them. Doing this gives you a control over the data, you can make it read only or write only.

e.g.

**public** **class** Encap{

**private** String name;

**public** String getName(){

**return** name;

}

**public** **void** setName(String name){

**this**.name=name;

}

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

Encap s=**new** Encap();

s.setName("vijay");

System.***out***.println(s.getName());

}}

***Polymorphism*** is the capability of a method to do different things based on the objects that it is acting upon. It allows defining one interface and have multiple implementations. Polymorphism can be enforced in Java in two ways:

Q3(Part 1)

1. ***Method Overloading*** – In a class, if we have two or more methods with the same name, we must have different arguments. This is useful because, when a user does not have an idea and he enters whatever at the compile time the program can use the corresponding method to the input.(compile time)

e.g.

public class method{

void test(){

}

void test(int num){

}

void test(Boolean a){

}

public static void main(String[] Args){

method m = new method();

m.test();

m.test(10);

m.test(true);

}

}

Q3(Part2)

1. ***Method Overriding*** – In a parent and child class relationship, when there is same name but different definition it is called method overriding.(runtime)

e.g.

public class A{

void test(){

syso(“ To cast vote age is 18”);

}

public class B Extends A{

void test(){

syso(“To cast vote age is 20”);

}

}

public static void main(String[] Args){

A a = new A();

a.test(); //output (“To cast vote age is 18”)

B b = new B();

b.test(); //output (“To cast vote age is 20”)

}

}

Q4.Why is **string immutable**?

Ans. An immutable class is a class whose instances cannot be modified. All information is initialized when the instance is created and it cannot be changed. Why String is immutable in Java depends on several reasons:

1. Requirement of String pool – String pool is special area in the storage space. When a string is created and if the string already exists, the reference of existing string is returned instead of creating a new reference. If string is immutable, changing string with one reference will lead to the wrong value to other references.
2. Caching Hashcode- Hashcode is frequently used in Java. Being immutable guarantees that hashcode will always be the same.
3. Facilitating the use of other objects
4. Security – String is widely used as a parameter for network connection, opening files etc. If strings are mutable, a connection or file would be changed and can lead to a serious security threat.
5. Immutable objects are thread safe – Thread safe means objects that can shared among multiple threads freely.

Q5. What is the difference between String and String buffer?

Ans. 1. **String** is ***immutable***

Whereas, **String buffer** is ***mutable***

2. **String** is **slow** and consume more memory while concatenating too many strings because ***every time it*** ***creates a new instance***.

Whereas, **String buffer** is ***faster and consumes less memory*** while concatenation.

1. String class overrides the ***equals()*** method. So it can used to compare two strings,

Whereas, we cannot use ***equals()*** method while using string buffer.

Q6. What is the difference between Array and Array List?

Ans. 1. **Array** have ***specific size***, whereas **Array Lists** can ***grow***.

2. Performance – Performance of Array and Array list depends on the operation that is being performed. Array list has an initial size value, once it is exceeded it grows double which is time consuming, so in these kind of situations arrays perform better.

3. Primitives – **Array List** **CANNOT** contain ***primitive data types*** (such as int, float, double) it can only contain objects whereas arrays can contain both.

Q7. Difference between Hash Map and Hash Table?

Ans. 1. Synchronization and Thread safe – **Hash map** is **NOT** *synchronized* and **NOT** *thread safe*. **Hash Table IS** *synchronized* and *thread safe*.

2. Null keys and values – **Hash map** **ALLOWS** *one null key and multiple null values*, whereas, **Hash table** does **NOT** allow any null key or values.

3**. Hash map** is ***fast***, whereas **Hash table** is ***slow***.

4. **Hash map** can be ***traversed by iterator***, whereas, **Hash table** can ***traversed by iterator and enumerator***.

5. **Hash map** *inherits* ***Abstract Map*** whereas, **Hash table** *inherits* ***Dictionary class***.

Q8. What is a vector in Java?

Ans. Vectors (the java.util.Vector class) are commonly used instead of arrays, because they expand automatically when new data is added to them. Vector is synchronized

* **Create a Vector with default initial size**   
  Vector v = new Vector();
* **Create a Vector with an initial size**   
  Vector v = new Vector(300);

Q9. What is set in Java?

Ans. A set is collection that cannot contain duplicate elements. Set interface only contains methods inherited from collections.

Q10. What is an abstract class?

Ans. In a class, if there exists any unimplemented (or abstract) method it is called an abstract class. The abstract methods are defined in the child class.

e.g.

public abstract class demo{

void test(){

syso(“this is an implemented method”);

}

abstract void test1();

abstract void test2();

abstract void test3();

}

In the above example because there is at least one unimplemented method the above class becomes abstract.

Q11. What is an Interface?

Ans. An interface can have only abstract methods.

Q12.Why is Java platform independent?

Ans. Java is platform independent because program written in Java can be executed on any OS with any hardware.

For programs written in Java, the process is a bit different. Instead of directly being compiled into the machine code, Java source code is compiled into an intermediate code called byte code. The byte code is not specific to any operating system, in other words, it is platform independent.

Q13. What are access modifiers? Give example.

Ans. The access modifiers specify the accessibility of a data member, method, constructor or class. There are 4 types:

1. Private:-This access modifier is only accessible within class.

e.g.

class A{

private int a = 40;

private void message(){

syso(“this method is private”);

}

}

public class B{

public static void main(String[] Args){

A obj = new A(); //creating an object of class A.

Syso(obj.a); //It will throw compile time error since “a” is private.

obj.message(); // again compile time error

}

}

1. Default:- This type is only accessible within a package.

e.g.

package pack1;

public class A{

void message(){

syso(“this is class A”);

}

}

package pack2;

import pack1;

class B{

public static void main(String[] Args){

A obj = new A(); //compile time error

obj.message(); //compile time error

}

}

3. Protected:- This type is accessible within and outside package but only through inheritance. It can be applied to data member, method and constructor but not to a class.

e.g.

package pack1;

class A{

void message(){

syso(“this is class A”);

}

}

package pack2;

import pack1;

class B extends A{

public static void main(String[] Args){

A obj = new A();

obj.message(); //will print the message from class A.

}

}

4. Public :-This is accessible everywhere.

Q14.What are Java exceptions? Give example.

Ans. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program’s instruction, the application terminates abnormally.

The abnormal conditions that might end up in an exception are: divide by 0, array index out of bound.

The exceptions can be handled in Java using: try, catch and finally. We can also use throwable and throws.

e.g.

try{

int a = num/0; //statement that may cause an exception

}

catch(Arithematic e){

syso(“Error: Cannot divide number by 0”);

}

finally{

syso(“This block is always executed”);

}

Q15. What is the difference between throw and throwable.

Ans. Throwable is at the class level, **it** is a super class for all types of errors and exceptions in java, and throw is at method level.

Class A extends throwable{

public void add throws exception{

}

}

Q16. What is the difference between error and exception?

Ans. An error is an irrecoverable condition occurring at runtime like out of memory error. These kind of JVM errors cannot be handled at runtime. Exceptions are because of condition failures, which can be handled easily at runtime.

Exceptions are of two types:- checked(compile time) and unchecked(Runtime exception).

e.g. arithmetic, null pointer, stack overflow, array out of bound

Q17. What is the difference between Error, throwable and exception?

Ans. An error is an irrecoverable condition occurring at runtime like out of memory error. These kind of JVM errors cannot be handled at runtime.

Exceptions are because of condition failures, which can be handled easily at runtime.

Throwable is the super class of both error and exception.

public class JavaExceptions {

public static void main(String[] Args) {

int a = 10;

int b = 0;

try{

System.out.println(a/b);

}

catch(Throwable e){ //this will catch both error and exception

System.out.println(e);

}

/\*catch(Exception e){ //this will catch an exception

System.out.println(e);

}\*/

System.out.println("Thank you");

}

}

Q18. What are collection APIs? Give an example.

Ans. The Collection API is a set of classes and interfaces that support operation on collections of objects. These classes and interfaces are more flexible, more powerful, and more regular than the vectors, arrays, and hashtables if effectively replaced.

Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

Q19.What is the difference between final and finally?

Ans. In Java, **Final is a keyword**. It can be applied to a variable, a method and a class.

1. When a variable is declared final, it cannot be re-initialized.

e.g.

final int a = 10;

a = 20; //this will throw an error.

2. When a method is Final, it cannot be overrided.

e.g.

ClassA{

final void test(){

syso(“this is a final method”);

}

}

Class B extends A{

void test(){ //error

syso(“This is a overridden method”);

}

}

3. When a Class is declared final, it cannot be inherited.

e.g.

final class A{

}

Class B extends A{ //error

}

Whereas, **Finally is a block that is used to execute important code such as closing connection, stream etc.**

Java finally block is **always executed** whether exception is handled or not.

Java finally block must be followed by try or catch block.

Q20. Will java supports multiple inheritance?

Ans. Java does not support multiple inheritance. The reason why it does not support this is because it can cause Ambiguity in some scenarios, e.g. Diamond problem.

Diamond problem: Consider a class D, that extends class B and C which overrides a method in class A in their own different ways. Now, if D wants to use this particular method which method will it call (Ambiguity)?

To solve this problem and to be able to apply multiple inheritance in Java, Java provides something called interfaces. Interfaces can only contain abstract methods, which does not let them to give their implementation. So when a class implements multiple interfaces it can have its own implementation of the method without ambiguity.

e.g. of Interfaces and multiple inheritance

Interface A{

public void method();

}

Interface B{

public void method();

}

Class A implements A, B{

public void method(){

syso(“Multiple inheritance example”);

}

}

Q 21. What are the different types of interface? (Ans List, set, Queue)

Ans. 1. Set – It is a collection that cannot contain duplicate elements.

2. List – It is an ordered collection which can contain duplicate elements. The user of a List generally has precise control over where in the list each element is inserted and can access elements by their integer index (position).

1. Queue - a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations.

Q22. What are wrapper class? Give an example.

Ans. Wrapper classes in Java provides the mechanism to convert primitive into object and object into primitive. Example of wrapper classes in Java are:-

|  |  |
| --- | --- |
| Primitive | Wrapper Class |
| boolean | Boolean |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

Q23. What is boxing and unboxing in Java? Explain with an example.

Ans. **Boxing – *primitive to object***

e.g.

public class wrapper{

public static void main(String[] Args){

int a = 20;

Integer i = Integer.valueOf(a); //converting int into Integer

//autoboxing, now compiler will write Integer.valueOf(a) internally

Integer j = a;

syso(j);

}

}

**Unboxing – *Object to primitive***

e.g.

public class WrapperExample2{

public static void main(String args[]){

//Converting Integer to int

Integer a=new Integer(3);

int i=a.intValue();//converting Integer to int

//unboxing, now compiler will write a.intValue() internally

int j=a;

System.out.println(a+" "+i+" "+j);

}

}

Q24. Explain for each loop.

Ans. For each loop is used to traverse through an array or a collection.

Advantages:-

* It makes the code more readable.
* It eliminates the possibility of programming errors.

Syntax:-

for(<data type><variable> :array|collection)

e.g.

class ForEachEx{

public static void main(String args[]){

int arr[]={12,13,14,44};

for(int i:arr){

System.out.println(i);

}

}

}

Q25. What are iterators, explain with an example.

Ans. Java iterator is an interface belongs to collection framework allow us to traverse the collection and access the data element of collection without bothering the user about specific implementation of that collection it. Basically List and set collection provides the iterator. 

Q26. How do you access Private variables in different class?

Ans. The correct way to access a private variable from another class is with getter and setter methods.

Q27. Prepare for one java program to write on the board.

Ans. import java.util.Scanner;

public class Palindrome {

public static void main(String[] Args) {

String str;

String reverse = "";

System.out.println("Enter a string");

Scanner s = new Scanner(System.in);

str = s.nextLine();

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

reverse = reverse + str.charAt(i);

System.out.println(reverse);

if(str.equals(reverse)){

System.out.println("The string is a palindrome");

}

else{

System.out.println("The string is not a palindrome");

}

}}

Q28. What is Constructor Over loading?

Ans. Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.

Q29. Without using sync key word how do you perform synchronization?

Ans. You can make your code thread-safe by making all the data immutable, if there is no mutability, everything is thread-safe.

Q30. What is Super keyword? When and where do you use it?

Ans. Super keyword in java is a reference variable that is used to refer parent class object.

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

It plays important role in three places.

* At variable level
* At method level
* At constructor level