Java Concepts

1. DOES JAVA HAS DESTRUCTOR? HOW TO FORCE IT?

No. Because Java is a garbage collected language, you can not predict when an object will be destroyed, hence there is no direct equivalent of destructor.
System.gc()

2. HOW JAVA GARBAGE COLLECTION WORKS? IS IT GUARANTEED TO WORK?

Java garbage collection is the process by which Java programs perform automatic memory management. When Java programs run on the JVM, objects are created on the heap, once an object is no longer referenced and therefore is not reachable by the code, the garbage collector finds these unused objects and deletes them to free up memory.
No.In most cases, if there is insufficient memory remaining to satisfy the amount needed for a new object, then the garbage collector will attempt to reclaim as much memory as possible by releasing memory. However, it is perfectly possible for a developer to mistakenly create objects which never go out of scope, thus consuming more and more memory until all **heap** is exhausted.

3. EXPLAIN MUTEX

we use mutex to protect critical section and thus prevent race condition. Each time only one process will be able to have the key and proceed with their work. the we use acquire() function to acquire the lock before entering a critical section, and release() function to releases the lock when it exits the critical section. Mutex has a boolean variable available, it indicates if the lock is available or not. If the lock is available, and a call to acquire() succeeds, and the lock is then considered unavailable. A process that attempts to acquire an unavailable lock is blocked until the lock is released.

```
acquire() {
    while (!available)
       ; /* busy wait */
    available = false;;
}
```

```
do {
    acquire lock
        critical section
    release lock
        remainder section
} while (true);
```

```
release() {
   available = true;
}
```

4. EXPLAIN SEMAPHORE

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() and signal().
Because synch is initialized to 0, P_2 will execute S_2 only after P_1 has invoked signal(synch), which is after statement S_1 has been executed.

```
wait(S) {
    while (S <= 0)
       ; // busy wait
    S--;
}
```

```
signal(S) {
    S++;
}
```

```
S1;
signal(synch);
```

```
wait(synch);
S2;
```

In conclusion, mutex is locking mechanism, semaphore is signaling mechanism.

5. "==" VS EQUALS():

In general both equals() and “==” operator in Java are used to compare objects to check equality but here are some of the differences between the two:We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if

both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

6. WHAT HAPPENS WHEN COMPILING AND RUNNING JAVA CODE?

In Java, programs are not compiled into executable files; they are compiled into bytecode, which the JVM then executes at runtime. Java source code is compiled into bytecode when we use the `javac` compiler. The bytecode gets saved on the disk with the file extension.class,When the program is to be run, the bytecode is converted, using the just-in-time (JIT) compiler. The result is machine code which is then fed to the memory and is executed.

7. HASHTABLE V.S HASHMAP

there are several differences btw hashmap & hashtable.
1.hashtable is synchronized, whereas hashmap is not. This makes hashmap better for non-threaded applications, as unsynchronized objectes typically perform better than synchronized ones.HashMap can be synchronized by Map m = Collections.synchronizeMap(hashMap)
2.hashtable does not allow null keys or values. hashmap allows one null key and any number of null values.

8. POLYMORPHISM，CHILD OVERIDES PARENT，CALLING METHOD EXISTS IN CHILD BUT NOT IN PARENT.

A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

Overriding is a feature that allows a child class to provide a specific implementation of a method that is already provided by its parent classes. When a method in a subclass has the same name, same parameters or signature and same return type, as a method in its super-class, then the method in the subclass is said to override the method in the super-class.Method overriding is one of the way by which java achieve Run Time Polymorphism.The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

Use downcasting when we want to access specific behaviors of a subtype.

9. GENERIC TYPE

The idea is to allow type (Integer, String, … etc and user defined types) to be a parameter to methods, classes and interfaces.

10. WHY JAVA GENERIC

type parameters provide a way for you to re-use the same code with different inputs.
Code that uses generics has many benefits over non-generic code:
1.Stronger type checks at compile time.
A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
2.Elimination of casts.
3.Enabling programmers to implement generic algorithms.

10. WHAT IS FINALIZE()

The java.lang.Object.finalize() is called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

11. LINKED LIST VS DYNAMIC ARRAY() - FOR DELETING ELEMENT IN MIDDLE

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

12. YOU HAVE SOME CLASSES - USE THEM ANOTHER PROJECT .JAR FILES AND IMPORT.

13. DIFFERENCE BETWEEN STACK MEMORY AND HEAP MEMORY

1. stack memory is used to store local variables and function call while heap memory is used to store objects in Java
Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM .
2.Stack frame access is easier than the heap frame as the stack have small region of memory and is cache friendly, but in case of heap frames which are dispersed throughout the memory so it cause more cache misses.

14. HOW TO CREATE THREAD

To create a new thread, your program will either extend Thread or implement the Runnable interface.
To implement Runnable interface, a class need only implement a single method called run( ), which is declared like this:

```
public void run( )
```

Inside run( ), we will define the code that constitutes the new thread. Example:

```
public class MyClass implements Runnable {
public void run(){
System.out.println("MyClass running");
   }
}
```

To execute the run() method by a thread, pass an instance of MyClass to a Thread in its constructor (A **constructor in Java** is a block of code similar to a method that's called when an instance of an object is created). Here is how that is done:

```
Thread t1 = new Thread(new MyClass ());
t1.start();
```

When the thread is started it will call the run() method of the MyClass instance instead of executing its own run() method. The above example would print out the text "**MyClass running** ".

Extending Java Thread

The second way to create a thread is to create a new class that extends Thread, then override the run() method and then to create an instance of that class. The run() method is what is executed by the thread after you call start(). Here is an example of creating a Java Thread subclass:

```
public class MyClass extends Thread {
     public void run(){
     System.out.println("MyClass running");
   }
}
```

To create and start the above thread:

```
MyClass t1 = new MyClass ();
T1.start();
```

When the run() method executes it will print out the text " **MyClass running** ".

So far, we have been using only two threads: the **main** thread and one **child** thread. However, our program can affect as many threads as it needs. Let's see how we can create multiple threads.

15. CAN JAVA CODE RUN IN DIFFERENT PLATFORM

Yes, JVM

16. ACCESS MODIFIER

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:
Default ,Private, Protected, Public
Default: having default access modifier are accessible only within the same package.
Private: The methods or data members declared as private are accessible only within the class in which they are declared. Any other class of same package will not be able to access these members.
Top level Classes or interface can not be declared as private because
protected: The methods or data members declared as protected are accessible within same package or sub classes in different package.
public: Classes, methods or data members which are declared as public are accessible from every where in the program. There is no restriction on the scope of a public data members.

17. WHY OOP

abstraction, encapsulation, inheritance, polymorphism

- OOP provides a clear modular structure for programs.
  - It is good for defining abstract data types.
  - Implementation details are hidden from other modules and other modules has a clearly defined interface.
  - It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
  - It implements real life scenario.
  - In OOP, programmer not only defines data types but also deals with operations applied for data structures.

18

```
class Sample {
foo1()
static foo2()
}
```

what's the difference between them, when you call it from the outside;

```
Sample sample1=new Sample();
sample1.foo();
Sample.foo2();  // static function is not related instance
```

19.EIGENVECTOR

$$A\vec{v} = \lambda\vec{v}$$

eigenvector do not change direction when a linear transformation is applied to them.