**Q1:**

**Implicit lock versus explicit lock:**

Explicit and implicit locks can control shared resources. Both are the same mechanism for memory synchronization, but explicit locks provide a more flexible and powerful interface.

Synchronized multiple locks can only be released in the reverse order of the order in which they were acquired (first acquired and then released). The lock is released as needed, and there is no such constraint. Explicit locks provide interruptible methods for acquiring locks. Explicitly provide an attempt to acquire a lock method. Provide more precise waiting and wake-up (use conditions).

**Class lock versus Object lock:**

Object-level locking: Every object in java has a unique lock. Whenever we are using synchronized keyword, then only lock concept will come in the picture. If a thread wants to execute synchronized method on the given object. First, it has to get lock of that object. Once thread got the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock.

Class-level locking: Every class in java has a unique lock which is nothing but class level lock. If a thread wants to execute a static synchronized method, then thread requires class level lock. Once a thread got the class level lock, then it is allowed to execute any static synchronized method of that class. Once method execution completes automatically thread releases the lock.

**Call Stack:**

The main function of the call stack is to save the return address of the call.The role of the stack in function calls: parameter passing, local variable allocation, saving the return address of the call, saving registers for recovery

**Stack Memory:**

Stack memory is a memory usage mechanism that allows the system memory to be used as temporary data storage that behaves as a first-in-last-out buffer. One of the essential elements of stack memory operation is a register called the Stack Pointer. The stack pointer indicates where the current stack memory location is, and is adjusted automatically each time a stack operation is carried out.

**Heap Space:**

Heap space in Java is used for dynamic memory allocation for Java objects and JRE classes at the runtime. It can be divided into Young generation and Old generation

**String Pool:**

As the name suggests, String Pool in java is a pool of Strings stored in Java Heap Memory. We know that String is special class in java and we can create String object using new operator as well as providing values in double quotes.

Example: String str0 = “a”; String str1 = “b”; String str2 = “c”;

**Deadlock, Starvation, Race condition:**

**Deadlock:**

A Deadlock is when two or more threads are blocked waiting to obtain locks that some other threads in the deadlock are holding. Deadlock can occur when multiple threads need the same locks, at the same time, but obtain them in different order. For instance, if thread-1 locks A, and tries to lock B, and thread-2 has already locked B, and tries to lock A, a deadlock arises. Thread-1 can never get B, and thread 2 can never get A. In addition, neither of them will ever know. They will remain blocked on each their object, A and B, forever. This situation is a deadlock.

Example:

Thread-1 locks A, waits for B

Thread-2 locks B, waits for A

**Starvation:**

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU. In starvation resources are continuously utilized by high priority processes.

Example:

Thread A with higher priority keeps on locking object C. Then thread B with lower priority keeps on waiting for the lock of object C which it will never get.

**Race condition:**

If two or more threads share an object, and more than one thread updates variables in that shared object, race conditions may occur.

Example:

When Thread A and Thread B share the same object which contains the constant count, they both add 1 to the constant. It should be incremented twice and had the original value +2. However, as the race condition occur, the result will be incremented only once instead.

**Q2:**

ThreadSafe Collection: Vector, HashTable, StringBuffer

NotThreadSafe Collection: ArrayList, LinkedList, HashMap, HashSet, TreeMap, TreeSet, StringBuilder

A)

ThreadSafe Collection:

a) public void add(int index, E element)：

The element element is added from the index position, and the subsequent elements are shifted back by one.

b) public Enumeration<V> elements():

Return an enumeration of the values in this HashTable。

c) public StringBuffer deleteCharAt(int index):

Delete the characters at the specified position, and then form the remaining content into a new string.

NotThreadSafe Collection:

a) removeFirst(E e)：

Delete header, get element and delete.

b) public Map.Entry<K,V> ceilingEntry(K key)：

Returns the element whose specified Key is greater than or equal to the minimum value, or null if none.

c) appendcodePoint(int cp)：

Append a code point and convert it to one or two code units and return this.

B)

Collections class: This class consists exclusively of static methods that operate on or return collections. It contains polymorphic algorithms that operate on collections, "wrappers", which return a new collection backed by a specified collection, and a few other odds and ends.

5 Methods:

addAll(Collection<? super T> c, T... elements)

asLifoQueue(Deque<T> deque)

checkedCollection(Collection<E> c, Class<E> type)

checkedList(List<E> list, Class<E> type)

copy(List<? super T> dest, List<? extends T> src)

**Q5:**

Class Person {

int pid;

String name;

// constructor, setters/getters

}

public class Driver {

int id = 23;

String pName = “Jon”;

Person p = null;

p = new Person (id, pName);

}

]

**Q6:**

A thread calling outer() will first lock the Lock instance. Then it will call inner(). Inside the inner() method the thread will again try to lock the Lock instance. This will fail (meaning the thread will be blocked), since the Lock instance was locked already in the outer() method.

**Q7:**

When the printJob method in PrinterQueue class is called, the object queueLock is locked by the current thread at the beginning of the method. So the other threads will wait until the unlock method is called. That’s the reason why the threads will execute one by one.