|  |  |
| --- | --- |
| **1.** | public class M1 {  public static void main(String[] args) {  System.out.println(1);  System.out.println(2);  System.out.println(3);  System.out.println(4);  System.out.println(5);  System.out.println(6);  }  } |
| |  | | --- | | A.  1  2  3  4  5  6 |  |  | | --- | | B.  1 2 3 4 5 6 | | | |
| **Correct Answer: A** | | |
| **2.** | by default how many threads are involving in the foreground execution? |
| |  | | --- | | A.  1 |  |  | | --- | | A.  2 |  |  | | --- | | C.  3 | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **3.** | In which package Thread class is available? |
| |  | | --- | | A.  java.threads |  |  | | --- | | B.  java.lang |  |  | | --- | | C.  java.io | | | |
| **Correct Answer: B** | | |
| **4.** | Thread is a concrete class. |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **5.** | We can create an Object to Thread class |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | |
| **Correct Answer: A** | | |
| **6.** | Which method of a Thread class we need to override to incorporate a thread task? |
| |  | | --- | | A.  start |  |  | | --- | | B.  task |  |  | | --- | | C.  run | | | |
| **Correct Answer: C** | | |

|  |  |  |
| --- | --- | --- |
| **7.** | Which method of a Thread class we need to call to execute a new thread in the separate thread execution context. | |
| |  | | --- | | A.  start |  |  | | --- | | B.  task |  |  | | --- | | C.  run | | | | |
| **Correct Answer: A** | | | |
| **8.** | Which method of a Thread class we need to call to register a new thread with the thread schedular. |
| |  | | --- | | A.  start |  |  | | --- | | B.  task |  |  | | --- | | C.  run | | | | |
| **Correct Answer: A** | | | |

|  |  |  |
| --- | --- | --- |
| **9.** | In which package Runnable interface is available? | |
| |  | | --- | | A.  java.lang |  |  | | --- | | B.  java.io |  |  | | --- | | C.  java.threads | | | | |
| **Correct Answer: A** | | | |
| **10.** | | How many methods available in the Runnable interface |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  3 |  |  | | --- | | D.  no methods | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **11.** | Runnable interface is a marker interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **12.** | Runnable interface is a functional interface |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **13.** | Which method available in the Runnable interface? |
| |  | | --- | | A.  start |  |  | | --- | | B.  thread |  |  | | --- | | C.  run | | | |
| **Correct Answer: C** | | |
| **14.** | Which method we have to implement while developing a subclass to Runnable interface? |
| |  | | --- | | A.  start |  |  | | --- | | B.  thread |  |  | | --- | | C.  run | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **15.** | How many ways are there to develop child threads in Java programming language? |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  3 |  |  | | --- | | D.  no methods | | | | |
| **Correct Answer: B** | | | |
| **16.** | In which method we have to implement thread task while developing a child thread by using Runnable interface? | |
| |  | | --- | | A.  start |  |  | | --- | | B.  thread |  |  | | --- | | C.  run | | | | |
| **Correct Answer: C** | | | |

|  |  |  |
| --- | --- | --- |
| **17.** | Which reference type we should suply to the constructor of Thread class while developing a thread by making use of Runnable interface? | |
| |  | | --- | | A.  ThreadLocal |  |  | | --- | | B.  Number |  |  | | --- | | C.  Runnable | | | | |
| **Correct Answer: C** | | | |
| **18.** | start() method is available in the Runnable interface. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **19.** | We can call start() method from a class which is subclass to Runnable interface. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **20.** | We can call start() method from a class which is subclass to Thread class. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **21.** | We can call start() method from a Thread class. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **22.** | What happens if main thread is calling run() method from the child thread instead of start() method. | |
| |  | | --- | | A.  Compilation error |  |  | | --- | | B.  Runtime Error |  |  | | --- | | C.  Compilation and Running both success and child thread will be allocated to execute run() method., |  |  | | --- | | D.  Compilation and Running both success and main thread will be executing run() method., | | | | |
| **Correct Answer: D** | | | |

|  |  |  |
| --- | --- | --- |
| **23.** | if main thread is calling run() method from the child thread instead of start() method, then child will be registring with thread schedular. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **24.** | is it possible to call run() method more than once in the same thread reference? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **25.** | is it possible to call start() method more than once in the same thread reference? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **26.** | what happens while start() method calling more than once in the same thread reference? | |
| |  | | --- | | A.  Compilation error |  |  | | --- | | B.  Runtime Error |  |  | | --- | | C.  executing more than one time. | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **27.** | what happens while run() method calling more than once in the same thread reference? |
| |  | | --- | | A.  Compilation error |  |  | | --- | | B.  Runtime Error |  |  | | --- | | C.  executing more than one time. | | | |
| **Correct Answer: C** | | |
| **28.** | Unhandled Exception object effects only to the current thread. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **29.** | Unhandled Exception object effects to all the running threads of current execution. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **30.** | If main thread recives unhandled exception, then main thread only terminates from the execution. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **31.** | If main thread recives unhandled exception, then all the treads of the current exection terminates. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **32.** | If child thread recives unhandled exception, then all the treads of the current exection terminates. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **33.** | If child thread recives unhandled exception, then main thread terminates from the execution. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **34.** | currentThread() is a static method in the Thread class. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **35.** | currentThread() is returning reference of a thread which executes currentThread() method. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **36.** | Which t1 refers main thread. (Assume A is a subclass to Thread and B is a subclass to Runnable) | |
| |  | | --- | | A.  A t1 = new A() |  |  | | --- | | B.  B t1 = new B() |  |  | | --- | | C.  Thread t1 = Thread.currentThread() |  |  | | --- | | D.  Thread t1 = B.currentThread() |  |  | | --- | | E.  Thread t1 = A.currentThread() | | | | |
| **Correct Answer: C** | | | |

|  |  |
| --- | --- |
| **37.** | Is it possible to define more than one Thread to A class, if A is subclass to Thread class. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **38.** | Is it possible to define more than one Thread to B class, if B is subclass to Runnable interface | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **39.** | How many minimum number of objects required to A class to define 2 threads. (Assume A is a subclass to Thread) |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  we cant define multiple threads to the same class | | | | |
| **Correct Answer: B** | | | |
| **40.** | How many minimum number of objects required to A class to define 2 threads. (Assume A is a subclass to Runnable) | |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  we cant define multiple threads to the same class | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **41.** | Every thread is having an unique id. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **42.** | what is the data type of id of a thread |
| |  | | --- | | A.  int |  |  | | --- | | B.  long |  |  | | --- | | C.  double |  |  | | --- | | D.  String | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **43.** | How to change a thread ID? |
| |  | | --- | | A.  set() |  |  | | --- | | B.  setId() |  |  | | --- | | C.  setThreadId() |  |  | | --- | | D.  we cant change | | | |
| **Correct Answer: D** | | |
| **44.** | How to read a thread ID? |
| |  | | --- | | A.  get() |  |  | | --- | | B.  getId() |  |  | | --- | | C.  getThreadId() |  |  | | --- | | D.  we cant read | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **45.** | What is the default name to main thread? |
| |  | | --- | | A.  main |  |  | | --- | | B.  initiator |  |  | | --- | | C.  parent |  |  | | --- | | D.  no name | | | |
| **Correct Answer: A** | | |
| **46.** | What is the default name to 1st child thread? |
| |  | | --- | | A.  main |  |  | | --- | | B.  thread-0 |  |  | | --- | | C.  thread-1 |  |  | | --- | | D.  thread-2 | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **47.** | What is the default name to the 2nd child thread? |
| |  | | --- | | A.  main |  |  | | --- | | B.  thread-0 |  |  | | --- | | C.  thread-1 |  |  | | --- | | D.  thread-2 | | | |
| **Correct Answer: C** | | |
| **48.** | Is it possible to change name to main thread? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **49.** | how to change default name of a thread? |
| |  | | --- | | A.  setName() |  |  | | --- | | B.  setThreadName() |  |  | | --- | | C.  we cant change | | | |
| **Correct Answer: A** | | |
| **50.** | is it possible to choose a name to child thread while creating? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **51.** | what is the miximum priority in case of threads. |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: C** | | |
| **52.** | what is the minimum priority in case of threads. |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **53.** | what is the normal priority in case of threads. |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: B** | | |
| **54.** | What is the value of Thread.MIN\_PRIORITY |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **55.** | What is the value of Thread.NORM\_PRIORITY |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: B** | | |
| **56.** | What is the value of Thread.MAX\_PRIORITY |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **57.** | What is the default priority of a main thread |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 | | | |
| **Correct Answer: B** | | |
| **58.** | What is the default priority of a child thread |
| |  | | --- | | A.  1 |  |  | | --- | | B.  5 |  |  | | --- | | C.  10 |  |  | | --- | | D.  its a priority of its parent thread. | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **59.** | what is the datatype of a priority |
| |  | | --- | | A.  int |  |  | | --- | | B.  long |  |  | | --- | | C.  double |  |  | | --- | | D.  String | | | |
| **Correct Answer: A** | | |
| **60.** | how to read the priority of a thread |
| |  | | --- | | A.  get() |  |  | | --- | | B.  getPriority() |  |  | | --- | | C.  getP() |  |  | | --- | | D.  getInt() | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **61.** | how to change the priority of a thread |
| |  | | --- | | A.  set() |  |  | | --- | | B.  setPriority() |  |  | | --- | | C.  setP() |  |  | | --- | | D.  setInt() | | | |
| **Correct Answer: B** | | |
| **62.** | is it possible to change the priority of a main thread |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **63.** | if child is depending on parent thread life, then child is called as? |
| |  | | --- | | A.  deamon |  |  | | --- | | B.  user | | | |
| **Correct Answer: A** | | |
| **64.** | if child is not depending on parent thread life, then child is called as? |
| |  | | --- | | A.  deamon |  |  | | --- | | B.  user | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **65.** | deamon is a thread which is depending on its parent. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **66.** | deamon is a thread which is not depending on its parent. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **67.** | user thread is a thread which is depending on its parent. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **68.** | user thread is a thread which is not depending on its parent. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **69.** | what is the defualt type of a thread in java |
| |  | | --- | | A.  deamon |  |  | | --- | | B.  user | | | |
| **Correct Answer: A** | | |
| **70.** | which type of a thread creating while developing with a Thread class |
| |  | | --- | | A.  deamon |  |  | | --- | | B.  user | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **71.** | which type of a thread creating while developing with a Runnable interface |
| |  | | --- | | A.  deamon |  |  | | --- | | B.  user | | | |
| **Correct Answer: A** | | |
| **72.** | how to read a deamon status? |
| |  | | --- | | A.  getDeamon() |  |  | | --- | | B.  isDeamon() | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **73.** | how to change a thread from deamon to user (or) from user to deamon? |
| |  | | --- | | A.  setDeamon() |  |  | | --- | | B.  changeDeamon() |  |  | | --- | | C.  we cant change | | | |
| **Correct Answer: A** | | |
| **74.** | When we can change a thread from deamon to user (or) from user to deamon? |
| |  | | --- | | A.  before start() |  |  | | --- | | B.  after start() |  |  | | --- | | C.  we cant change | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **75.** | is it possible to change a main thread as a deamon? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **76.** | sleep() method in Thread class is a static. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **77.** | what is the argument data type of a sleep method in Thread class? |
| |  | | --- | | A.  int |  |  | | --- | | B.  long |  |  | | --- | | C.  float |  |  | | --- | | D.  double | | | |
| **Correct Answer: B** | | |
| **78.** | which checked exception required for sleep method in Thread class? |
| |  | | --- | | A.  ArithmaticException |  |  | | --- | | B.  NumberFormatException |  |  | | --- | | C.  InterruptedException |  |  | | --- | | D.  double | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **79.** | which checked exception required for join method in Thread class? |
| |  | | --- | | A.  ArithmaticException |  |  | | --- | | B.  NumberFormatException |  |  | | --- | | C.  InterruptedException |  |  | | --- | | D.  double | | | |
| **Correct Answer: C** | | |
| **80.** | How to tell to parent to wait till child execution got over? |
| |  | | --- | | A.  through sleep() |  |  | | --- | | B.  through join() | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **81.** | is it possible to interrupt a thread which is under sleep? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **82.** | is it possible to interrupt a thread which is under join? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **83.** | whih method is used to interrupt while thread is under sleep? |
| |  | | --- | | A.  interrupt() |  |  | | --- | | B.  makeStop() |  |  | | --- | | C.  stopInterrupt() | | | |
| **Correct Answer: A** | | |
| **84.** | whih method is used to interrupt while thread is under join? |
| |  | | --- | | A.  interrupt() |  |  | | --- | | B.  makeStop() |  |  | | --- | | C.  stopInterrupt() | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **85.** | How many maximum locks will be for one object? |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  it depends on how many super classes are there? | | | |
| **Correct Answer: A** | | |
| **86.** | How many maximum locks will be for one class? |
| |  | | --- | | A.  1 |  |  | | --- | | B.  2 |  |  | | --- | | C.  it depends on how many super classes are there? | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **87.** | Which lock is required while accessing synchronized and non static method |
| |  | | --- | | A.  class lock |  |  | | --- | | B.  object lock |  |  | | --- | | C.  no lock is required | | | |
| **Correct Answer: B** | | |
| **88.** | Which lock is required while accessing synchronized and static method |
| |  | | --- | | A.  class lock |  |  | | --- | | B.  object lock |  |  | | --- | | C.  no lock is required | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **89.** | Which lock is required while accessing non synchronized and static method |
| |  | | --- | | A.  class lock |  |  | | --- | | B.  object lock |  |  | | --- | | C.  no lock is required | | | |
| **Correct Answer: C** | | |
| **90.** | Which lock is required while accessing non synchronized and non static method |
| |  | | --- | | A.  class lock |  |  | | --- | | B.  object lock |  |  | | --- | | C.  no lock is required | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **91.** | to access a synchronized and non static method on a1, thread should required \_\_\_\_ |
| |  | | --- | | A.  object lock of any object |  |  | | --- | | B.  object lock of an object which is refered by a1 |  |  | | --- | | C.  object lock of an object which is refered by thread reference | | | | |
| **Correct Answer: B** | | | |
| **92.** | Assume a1 is a type of A class and it is pointing to an object of A class. To access a synchronized and non static method on a1, thread should required \_\_\_\_ | |
| |  | | --- | | A.  object lock of any object |  |  | | --- | | B.  object lock of an object which is refered by a1 |  |  | | --- | | C.  object lock of an object which is refered by thread reference |  |  | | --- | | D.  class lock of A class. | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **93.** | Assume a1 is a type of A class and it is pointing to an object of A class. To access a synchronized and static method on a1, thread should required \_\_\_\_ |
| |  | | --- | | A.  object lock of any object |  |  | | --- | | B.  object lock of an object which is refered by a1 |  |  | | --- | | C.  object lock of an object which is refered by thread reference |  |  | | --- | | D.  class lock of A class. | | | | |
| **Correct Answer: D** | | | |
| **94.** | Assume a1 is a type of A class and it is pointing to an object of A class. Is it possible to access two threads any nonstatic and non synchronized method on a1 simultaniously? | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **95.** | Assume a1 is a type of A class and it is pointing to an object of A class. Is it possible to access two threads any nonstatic and non synchronized different methods on a1 simultaniously? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **96.** | Assume a1 is a type of A class and it is pointing to an object of A class. Is it possible to access two threads any synchronized and nonstatic method on a1 simultaniously? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **97.** | Assume a1 is a type of A class and it is pointing to an object of A class. Is it possible to access two threads any synchronized and nonstatic different methods on a1 simultaniously? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: B** | | |
| **98.** | Assume a1 is a type of A class and it is pointing to an object of A class and a2 is a type of A class and it is pointing to another object of same A class . Assume A class containing synchronized and non static test1() method. Is it possible, one thread accessing a1.test1() and another thread accessing a2.test1() simultaniously? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **99.** | Assume a1 is a type of A class and it is pointing to an object of A class and a2 is a type of A class and it is pointing to another object of same A class . Assume A class containing synchronized and non static test1() and test2() method2. Is it possible, one thread accessing a1.test1() and another thread accessing a2.test2() simultaniously? | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **100.** | | Assume a1 is a type of A class and it is pointing to an object of A class. Assume A class containing synchronized and non static test1() and test2() method2. Is it possible, one thread accessing a1.test1() and another thread accessing a1.test2() simultaniously? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **101.** | Assume A class containing static and synchronized test1() method. Is it possible, two threads accessing A.test1() simultaniously? |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **102.** | Assume A class containing static and synchronized test1() and test2() methods. Is it possible, one thread accessing A.test1() and another thread accessing A.test2() simultaniously? | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |

|  |  |  |
| --- | --- | --- |
| **103.** | while thread is entering into synchronized and non static method on a1 pointing object, then it should required object lock of an object which is refered by a1. | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **104.** | If thread required object lock of current object to enter into synchronized block, then what could be the mutext to the synchronized block? |
| |  | | --- | | A.  Runnable reference |  |  | | --- | | B.  Thread reference |  |  | | --- | | C.  this | | | | |
| **Correct Answer: C** | | | |

|  |  |  |
| --- | --- | --- |
| **105.** | if t1 thread holding object lock of a1 and looking for a2 object lock. and if t2 thread holding object lock a2 and looking for a1 object lock. Then which thread will execute first. | |
| |  | | --- | | A.  t1 |  |  | | --- | | B.  t2 |  |  | | --- | | C.  dead lock | | | | |
| **Correct Answer: C** | | | |
| **106.** | In which class wait() method is developed? |
| |  | | --- | | A.  Thread |  |  | | --- | | B.  Object |  |  | | --- | | C.  ThreadLocal | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **107.** | In which class notify() method is developed? |
| |  | | --- | | A.  Thread |  |  | | --- | | B.  Object |  |  | | --- | | C.  ThreadLocal | | | |
| **Correct Answer: B** | | |
| **108.** | In which class notifyAll() method is developed? |
| |  | | --- | | A.  Thread |  |  | | --- | | B.  Object |  |  | | --- | | C.  ThreadLocal | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **109.** | String class having wait() method. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **110.** | Thread class having notify() method. |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **111.** | Thread should required object lock of t1 to call t1.wait() |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |
| **112.** | Thread should required object lock of t1 to call t1.notify() |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **113.** | Thread should required object lock of t2 to call t1.notifyAll() |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |
| **114.** | Thread having object lock of t1 and trying to call t2.wait(). t1 and t2 are refering to different objects. Now thread will go to waiting state or not? | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **115.** | which checked exception required to call wait() method? |
| |  | | --- | | A.  ClassNotFoundException |  |  | | --- | | B.  SQLException |  |  | | --- | | C.  InterruptedException | | | | |
| **Correct Answer: C** | | | |
| **116.** | main thread is about to call a1.wait() in the main method by choosing synchronized block. What could be the mutext for the synchronized block? | |
| |  | | --- | | A.  this |  |  | | --- | | B.  main thread reference |  |  | | --- | | C.  a1 | | | | |
| **Correct Answer: C** | | | |

|  |  |
| --- | --- |
| **117.** | child thread is about to call wait() in the run method. by choosing synchronized block. What could be the mutext for the synchronized block? |
| |  | | --- | | A.  this |  |  | | --- | | B.  main thread reference |  |  | | --- | | C.  a1 | | | |
| **Correct Answer: A** | | |
| **118.** | main thread is about to call a1.notify() in the main method by choosing synchronized block. What could be the mutext for the synchronized block? |
| |  | | --- | | A.  this |  |  | | --- | | B.  main thread reference |  |  | | --- | | C.  a1 | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **119.** | child thread is about to call notifyAll() in the run method. by choosing synchronized block. What could be the mutext for the synchronized block? |
| |  | | --- | | A.  this |  |  | | --- | | B.  main thread reference |  |  | | --- | | C.  a1 | | | | |
| **Correct Answer: A** | | | |
| **120.** | there are two objects. t1 is refering to one object and t2 is refering to another object. child1 went into waiting by calling wait on t1. how child1 can be released from the waiting? | |
| |  | | --- | | A.  some one has to call t1.notify() |  |  | | --- | | B.  some one has to call t2.notifyAll() | | | | |
| **Correct Answer: A** | | | |

|  |  |  |
| --- | --- | --- |
| **121.** | there are two objects. t1 is refering to one object and t2 is refering to another object. some child threads went into waiting by calling wait on t1. how all child threads can be released from the waiting? | |
| |  | | --- | | A.  some one has to call t1.notify() |  |  | | --- | | B.  some one has to call t2.notifyAll() |  |  | | --- | | C.  some one has to call t1.notifyAll() | | | | |
| **Correct Answer: A** | | | |
| **122.** | while thread is going to wait stage, it relases owned lock. |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **123.** | while thread is calling join, it relases owned lock. |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | |
| **Correct Answer: A** | | |
| **124.** | while thread is calling sleep, it relases owned lock. |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **125.** | ThreadLocal maintains a value local to a perticular thread. |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | | |
| **Correct Answer: A** | | | |
| **126.** | if same ThreadLocal object is using by two different threads and first thread set a value as 10 and 2nd thread not set a value yet. Then what could be the value while reading by 2nd thread. | |
| |  | | --- | | A.  10 |  |  | | --- | | B.  0 |  |  | | --- | | C.  null |  |  | | --- | | D.  exception | | | | |
| **Correct Answer: C** | | | |

|  |  |  |
| --- | --- | --- |
| **127.** | is it possibe to stop all the threads which are under one group by using group reference? | |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | | |
| **Correct Answer: A** | | | |
| **128.** | ThreadGroup is used for grouping any number of and any type of threads. |
| |  | | --- | | A.  true |  |  | | --- | | B.  false | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **129.** | what is the state of a thread after calling sleep method. |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  NEW |  |  | | --- | | E.  BLOCKED | | | |
| **Correct Answer: B** | | |
| **130.** | what is the state of a thread after calling wait method. |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  NEW |  |  | | --- | | E.  BLOCKED | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **131.** | what is the state of a thread after calling join method. |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  NEW |  |  | | --- | | E.  BLOCKED | | | | |
| **Correct Answer: A** | | | |
| **132.** | what is the state of a thread after calling start method while executing run method.. | |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  NEW |  |  | | --- | | E.  BLOCKED | | | | |
| **Correct Answer: C** | | | |

|  |  |
| --- | --- |
| **133.** | what is the state of a thread before calling start method |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  NEW |  |  | | --- | | E.  BLOCKED | | | |
| **Correct Answer: D** | | |
| **134.** | what is the state of a thread while it is in dead lock |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  NEW |  |  | | --- | | E.  BLOCKED | | | |
| **Correct Answer: E** | | |

|  |  |
| --- | --- |
| **135.** | what is the state of a thread once its execution got over |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  TIMED\_WAITING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  TERMINATED |  |  | | --- | | E.  BLOCKED | | | |
| **Correct Answer: D** | | |
| **136.** | yield() is used for requesting current thread to give a chance to another threads |
| |  | | --- | | A.  yes |  |  | | --- | | B.  no | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **137.** | public class M2 {  public static void main(String[] args) {  for(int i = 1; i <= 1000; i++)  {  System.out.println("first loop :" + i);  }  for(int i = 2000; i <= 3000; i++)  {  System.out.println("second loop :" + i);  }  for(int i = 4000; i <= 5000; i++)  {  System.out.println("third loop :" + i);  }  }  }  /\* Guess what could be the output \*/ |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **138.** | class A extends Thread  {  @Override  public void run() {  for(int i = 2000; i <= 3000; i++)  {  System.out.println("second loop :" + i);  }  }  }  class B extends Thread  {  @Override  public void run() {  for(int i = 4000; i <= 5000; i++)  {  System.out.println("third loop :" + i);  }  }  }  public class M3 {  public static void main(String[] args) {  A a1 = new A();  a1.start();  B b1 = new B();  b1.start();  for(int i = 1; i <= 1000; i++)  {  System.out.println("first loop :" + i);  }  }  }  /\* Guess what could be the output \*/ |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **139.** | class C implements Runnable  {  @Override  public void run() {  for(int i = 2000; i <= 3000; i++)  {  System.out.println("second loop :" + i);  }  }  }  class D implements Runnable  {  @Override  public void run() {  for(int i = 4000; i <= 5000; i++)  {  System.out.println("third loop :" + i);  }  }  }  public class M4 {  public static void main(String[] args) {  C c1 = new C();  Thread t1 = new Thread(c1);  t1.start();    D d1 = new D();  Thread t2 = new Thread(d1);  t2.start();  for(int i = 1; i <= 1000; i++)  {  System.out.println("first loop :" + i);  }  }  } |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | | |
| **Correct Answer: B** | | | |
| **140.** | class E extends Thread  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(getName() + ":" + i);  }  }  }  public class M5 {  public static void main(String[] args) {  E e1 = new E();  e1.start();  //e1.start();    E e2 = new E();  e2.start();  for(int i = 5000; i <= 6000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }  }  } | |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **141.** | class F implements Runnable  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }  }  }  public class M6 {  public static void main(String[] args) {  F f1 = new F();  Thread t1 = new Thread(f1);  t1.start();  //t1.start();    Thread t2 = new Thread(f1);  t2.start();  for(int i = 5000; i <= 6000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }  }  } |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: B** | | |
| **142.** | class G extends Thread  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }  }  }  public class M7 {  public static void main(String[] args) {  G g1 = new G();  //g1.start();  g1.run();  for(int i = 2000; i <= 3000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }    }  } |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **143.** | class H implements Runnable  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }  }  }  public class M8 {  public static void main(String[] args) {  H h1 = new H();  Thread t1 = new Thread(h1);  t1.start();  //h1.run();  //t1.run();  for(int i = 2000; i <= 3000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }    }  } | |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | | |
| **Correct Answer: B** | | | |
| **144.** | public class M9 {  public static void main(String[] args) {  Thread t1 = Thread.currentThread();  System.out.println(t1.getName());  System.out.println(t1.isDaemon());  System.out.println(t1.getPriority());  System.out.println(t1.getId());  }  } |
| |  | | --- | | A.  main  false  5  1 |  |  | | --- | | B.  main  true  5  1 |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  None |  |  | | --- | | E.  Runtime Error | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **145.** | class J extends Thread  {  @Override  public void run() {  System.out.println(getName());  System.out.println(isDaemon());  System.out.println(getPriority());  System.out.println(getId());  }  }  public class M10 {  public static void main(String[] args) {  J ref = new J();  ref.start();  }  } |
| |  | | --- | | A.  Thread-0  true  5  10 |  |  | | --- | | B.  Thread-0  false  5  10 |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  None |  |  | | --- | | E.  Runtime Error | | | |
| **Correct Answer: B** | | |
| **146.** | public class M11 {  public static void main(String[] args) {  Thread t1 = Thread.currentThread();  System.out.println(t1.getName());  t1.setName("initiator");  System.out.println(t1.getName());  }  } |
| |  | | --- | | A.  initiator  main |  |  | | --- | | B.  Thread  initiator |  |  | | --- | | C.  main  initiator |  |  | | --- | | D.  Compilation Error |  |  | | --- | | E.  None | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **147.** | public class M12 {  public static void main(String[] args) {  Thread t1 = Thread.currentThread();  System.out.println(t1.isDaemon());  t1.setDaemon(true);  System.out.println(t1.isDaemon());  }  } |
| |  | | --- | | A.  false  true |  |  | | --- | | B.  true  false |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  RunTime Exception |  |  | | --- | | E.  None | | | |
| **Correct Answer: D** | | |
| **148.** | public class M13 {  public static void main(String[] args) {  Thread t1 = Thread.currentThread();  System.out.println(t1.getPriority());  t1.setPriority(11);  System.out.println(t1.getPriority());  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  5  Exception |  |  | | --- | | C.  5  11 |  |  | | --- | | D.  None | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **149.** | class K extends Thread  {  @Override  public void run() {  System.out.println("child thread");  }  }  public class M14 {  public static void main(String[] args) {  K k1 = new K();  k1.start();  System.out.println(k1.getName());  k1.setName("first child");  System.out.println(k1.getName());  }  } |
| |  | | --- | | A.  Thread-0  first child  child thread |  |  | | --- | | B.  Thread-0  child thread  first child |  |  | | --- | | C.  Compile Time Error | | | |
| **Correct Answer: A** | | |
| **150.** | class L implements Runnable  {  @Override  public void run() {  System.out.println("child thread");  }  }  public class M15 {  public static void main(String[] args) {  L obj = new L();  Thread t1 = new Thread(obj);  t1.start();  System.out.println(t1.getName());  t1.setName("first child");  System.out.println(t1.getName());  }  } |
| |  | | --- | | B.  Thread-0  first child  child thread |  |  | | --- | | B.  Thread-0  child thread  first child |  |  | | --- | | C.  Compile Time Error | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **151.** | class M extends Thread  {  M(String name)  {  super(name);  }  @Override  public void run() {  System.out.println("child thread");  }  }  public class M16 {  public static void main(String[] args) {  M m1 = new M("my first thread");  m1.start();  System.out.println(m1.getName());  m1.setName("first child");  System.out.println(m1.getName());  }  } |
| |  | | --- | | A.  my first thread  child thread  first child |  |  | | --- | | B.  my first thread  first child  child thread |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  RunTime Exception | | | |
| **Correct Answer: A** | | |
| **152.** | class N implements Runnable  {  @Override  public void run() {  System.out.println("child thread");  }  }  public class M17 {  public static void main(String[] args) {  N obj = new N();  Thread t1 = new Thread(obj, "my first thread");  t1.start();  System.out.println(t1.getName());  t1.setName("first child");  System.out.println(t1.getName());  }  } |
| |  | | --- | | A.  my first thread  child thread  first child |  |  | | --- | | B.  my first thread  first child  child thread |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  RunTime Exception | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **153.** | class O extends Thread  {  @Override  public void run() {  System.out.println("from o thread:" + getPriority());  }  }  class P extends Thread  {  @Override  public void run() {  System.out.println("from p thread:" + getPriority());  setPriority(7);  O th1 = new O();  th1.start();  }  }  public class M18 {  public static void main(String[] args) {  Thread.currentThread().setPriority(10);  P p1 = new P();  p1.start();  }  } |
| |  | | --- | | A.  from o thread:7  from p thread:10 |  |  | | --- | | B.  from p thread:10  from o thread:7 |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  RunTime Exception | | | |
| **Correct Answer: B** | | |
| **154.** | class Q extends Thread  {  @Override  public void run() {  for(int i = 0; i < 1000; i++)  {  System.out.println(i);  }  }  }  public class M19 {  public static void main(String[] args) {  Q q1 = new Q();  q1.start();  System.out.println("done");  }  }  /\* Guess what could be the output\*/ |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **155.** | class R extends Thread  {  @Override  public void run() {  for(int i = 0; i < 1000; i++)  {  System.out.println(i);  }  }  }  public class M20 {  public static void main(String[] args) {  R r1 = new R();  r1.setDaemon(true);  r1.start();  System.out.println("done");  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  Runtime Exception |  |  | | --- | | C.  done |  |  | | --- | | D.  None | | | |
| **Correct Answer: C** | | |
| **156.** | class S extends Thread  {  @Override  public void run() {  for(int i = 0; i < 1000; i++)  {  System.out.println(i);  }  }  }  public class M21 {  public static void main(String[] args) {  S s1 = new S();  s1.start();  try  {  s1.join();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("done");  }  }  /\* Guess What could be the output\*/ |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **157.** | public class M22 {  public static void main(String[] args) {  System.out.println("main begin");  for(int i = 1; i <= 10; i++)  {  System.out.println(i);  try  {  Thread.sleep(1000);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  System.out.println("main end");  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  Runtime Exception |  |  | | --- | | C.  main begin  1  2  3  4  5  6  7  8  9  10  main end |  |  | | --- | | D.  1  main begin  2  3  4  5  6  7  8  9  10  main end | | | |
| **Correct Answer: C** | | |
| **158.** | public class M23 {  public static void main(String[] args) {  System.out.println("main begin");  for(int i = 1; i <= 10; i++)  {  System.out.println(i);  try  {  Thread.sleep(2000, 500);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  System.out.println("main end");  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  Runtime Exception |  |  | | --- | | C.  main begin  1  2  3  4  5  6  7  8  9  main end  10 |  |  | | --- | | D.  1  main begin  2  3  4  5  6  7  8  9  10  main end |  |  | | --- | | D.  main begin  1  2  3  4  5  6  7  8  9  10  main end | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **159.** | class T extends Thread  {  @Override  public void run() {  System.out.println("run begin");  for(int i = 100; i <= 110; i++)  {  System.out.println(i);  try  {  sleep(5000, 500);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  System.out.println("run end");  }  }  public class M24 {  public static void main(String[] args) {  T obj = new T();  obj.start();  System.out.println("main begin");  for(int i = 1; i <= 10; i++)  {  System.out.println(i);  try  {  Thread.sleep(2000, 500);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  System.out.println("main end");  }  } |
| |  | | --- | | A.  main begin  1  run begin  100  2  3  101  4  5  102  6  7  8  103  9  10  104  main end  105  106  107  108  109  110  run end |  |  | | --- | | B.  main begin  run begin  1  100  2  3  101  4  5  102  6  7  8  103  9  10  104  105  106  107  108  109  110  run end  main end |  |  | | --- | | C.  Compile Time Error |  |  | | --- | | D.  RunTime Exception | | | |
| **Correct Answer: A** | | |
| **160.** | class U extends Thread  {  @Override  public void run() {  System.out.println("run begin");  for(int i = 100; i <= 110; i++)  {  System.out.println(i);  Util.sleep(5000);  }  System.out.println("run end");  }  }  public class M25 {  public static void main(String[] args) {  U obj = new U();  obj.start();  System.out.println("main begin");  for(int i = 1; i <= 10; i++)  {  System.out.println(i);  Util.sleep(2000);  }  System.out.println("main end");  }  }  class Util {  public static void sleep(long millis)  {  try  {  Thread.sleep(millis);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  } |
| |  | | --- | | A.  main begin  run begin  1  100  2  3  101  4  5  102  6  7  8  103  9  10  104  105  106  107  108  109  110  run end  main end |  |  | | --- | | B.  main begin  1  run begin  100  2  3  101  4  5  102  6  7  8  103  9  10  104  main end  run end  105  106  107  108  109  110 |  |  | | --- | | C.  main begin  1  run begin  100  2  3  101  4  5  102  6  7  8  103  9  10  104  main end  105  106  107  108  109  110  run end |  |  | | --- | | D.  Compilation Error |  |  | | --- | | E.  Runtime Error | | | |
| **Correct Answer: C** | | |

|  |  |
| --- | --- |
| **161.** | class V extends Thread  {  @Override  public void run() {  System.out.println("run begin");  Util.sleep(10000);  System.out.println("run end");  }  }  public class M26 {  public static void main(String[] args) {  System.out.println("main begin");  V obj = new V();  obj.start();  Util.sleep(2000);  obj.interrupt();  System.out.println("main end");  }  }  class Util {  public static void sleep(long millis)  {  try  {  Thread.sleep(millis);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  Runtime Exception | | | |
| **Correct Answer: B** | | |
| **162.** | public class M27 {  public static void main(String[] args) {  System.out.println(1);  int i = 10 / 0;  System.out.println(2);  }  } |
| |  | | --- | | A.  Compilation Error |  |  | | --- | | B.  Runtime Exception |  |  | | --- | | C.  1  2 | | | |
| **Correct Answer: B** | | |

|  |  |
| --- | --- |
| **163.** | class W extends Thread  {  @Override  public void run() {  System.out.println("run begin");  int i = 10 / 0;  System.out.println("run end");  }  }  public class M28 {  public static void main(String[] args) {  System.out.println(1);  W obj = new W();  obj.start();  System.out.println(2);  }  } |
| |  | | --- | | A.  1  2  run begin  Exception |  |  | | --- | | B.  1  run begin  run end  2 |  |  | | --- | | C.  run begin  run end  1  2 |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **164.** | class X extends Thread  {  X()  {  start();  }  @Override  public void run() {  System.out.println("run begin");  System.out.println("run end");  }  }  public class M29 {  public static void main(String[] args) {  System.out.println(1);  X obj = new X();  System.out.println(2);  }  } |
| |  | | --- | | A.  1  2  run begin  run end |  |  | | --- | | B.  1  run begin  run end  2 |  |  | | --- | | C.  run begin  run end  1  2 |  |  | | --- | | D.  Compilation Error |  |  | | --- | | E.  Runtime Error | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **165.** | class Y extends Thread  {  Y()  {  start();  }  @Override  public void run() {  System.out.println("run begin");  System.out.println("run end");  }  }  public class M30 {  public static void main(String[] args) {  System.out.println(1);  Y obj = new Y();  obj.start();  System.out.println(2);  }  } |
| |  | | --- | | A.  1  run begin  run end  2 |  |  | | --- | | B.  1  run begin  run end  Exception |  |  | | --- | | C.  Compile Time Error | | | |
| **Correct Answer: B** | | |
| **166.** | class Z extends Thread  {  Z()  {  start();  }  @Override  public void run() {  System.out.println("run begin");  start();  System.out.println("run end");  }  }  public class M31 {  public static void main(String[] args) {  System.out.println(1);  Z obj = new Z();  System.out.println(2);  }  } |
| |  | | --- | | A.  1  2  run begin  run end |  |  | | --- | | B.  1  run begin  run end  2 |  |  | | --- | | C.  run begin  run end  1  2 |  |  | | --- | | D.  1  2  run begin  Exception | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **167.** | package com.lara.pack2;  class Test  {  int i;  }  class Util  {  static void sleep(long millis)  {  try  {  Thread.sleep(millis);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  }  class A extends Thread  {  Test t1;  A(Test t1)  {  this.t1 = t1;  }  @Override  public void run() {  System.out.println("a:" + t1.i);  t1.i = 10;  Util.sleep(500);  System.out.println("b:" + t1.i);  t1.i = 20;  Util.sleep(500);  System.out.println("c:" + t1.i);  t1.i = 30;  }  }  class B extends Thread  {  Test t1;  B(Test t1)  {  this.t1 = t1;  }  @Override  public void run() {  System.out.println("d:" + t1.i);  t1.i = 40;  Util.sleep(500);  System.out.println("e:" + t1.i);  t1.i = 50;  Util.sleep(500);  System.out.println("f:" + t1.i);  t1.i = 60;  }  }  public class M1 {  public static void main(String[] args) {  Test t1 = new Test();  t1.i = 70;  A a1 = new A(t1);  a1.start();  Util.sleep(250);  B b1 = new B(t1);  b1.start();  Util.sleep(20000);  System.out.println("g:" + t1.i);  }  } |
| |  | | --- | | A.  a:70  d:10  b:40  e:20  c:50  f:30  g:60 |  |  | | --- | | B.  a:70  b:40  d:10  e:20  g:60  c:50  f:30 |  |  | | --- | | C.  Compilation error |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |
| **168.** | package com.lara.pack3;  class Util  {  static void sleep(long millis)  {  try  {  Thread.sleep(millis);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  }  class A extends Thread  {  ThreadLocal t1;  A(ThreadLocal t1)  {  this.t1 = t1;  }  @Override  public void run() {  System.out.println("a:" + t1.get());  t1.set(10);  Util.sleep(500);  System.out.println("b:" + t1.get());  t1.set(20);  Util.sleep(500);  System.out.println("c:" + t1.get());  t1.set(30);  }  }  class B extends Thread  {  ThreadLocal t1;  B(ThreadLocal t1)  {  this.t1 = t1;  }  @Override  public void run() {  System.out.println("d:" + t1.get());  t1.set(40);  Util.sleep(500);  System.out.println("e:" + t1.get());  t1.set(50);  Util.sleep(500);  System.out.println("f:" + t1.get());  t1.set(60);  }  }  public class M1 {  public static void main(String[] args) {  ThreadLocal t1 = new ThreadLocal();  t1.set(70);  A a1 = new A(t1);  a1.start();  Util.sleep(250);  B b1 = new B(t1);  b1.start();  Util.sleep(20000);  System.out.println("g:" + t1.get());  }  } |
| |  | | --- | | A.  a:70  d:10  b:40  e:20  c:50  f:30  g:60 |  |  | | --- | | B.  a:null  d:null  b:10  e:40  c:20  f:50  g:70 |  |  | | --- | | C.  Compilation error | | | |
| **Correct Answer: B** | | |

|  |  |  |
| --- | --- | --- |
| **169.** | package com.lara.pack4;  class A extends Thread  {  A(ThreadGroup tg, String name)  {  super(tg, name);  }  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(getName() + ":" + i);  }  }  }  class B extends Thread  {  B(ThreadGroup tg, String name)  {  super(tg, name);  }    @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(getName() + ":" + i);  }  }  }  class C implements Runnable  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(Thread.currentThread().getName() + ":" + i);  }  }  }  public class M1 {  public static void main(String[] args) {  ThreadGroup tg = new ThreadGroup("myFirstGroup");  A a1 = new A(tg, "firstThread");  A a2 = new A(tg, "secondThread");  B b1 = new B(tg, "thirdThread");  B b2 = new B(tg, "fourthThread");    C c1 = new C();  Thread t1 = new Thread(tg, c1, "5thThread");  Thread t2 = new Thread(tg, c1, "6thThread");    a1.start();  a2.start();  b1.start();  b2.start();  t1.start();  t2.start();    tg.stop();  }  }  // Check which option is possible for this program | |
| |  | | --- | | A.  We will get the output |  |  | | --- | | B.  we won't get output |  |  | | --- | | C.  Both a and b possible |  |  | | --- | | D.  None | | | | |
| **Correct Answer: C** | | | |
| **170.** | package com.lara.pack5;  class Employee  {  //several members  }  class A extends Thread  {  @Override  public void run() {  // TODO Auto-generated method stub  super.run();  }  }  class B extends Employee implements Runnable  {  @Override  public void run() {  // TODO Auto-generated method stub    }  }  public class M1 {  public static void main(String[] args) {  A a1 = new A();  a1.start();  A a2 = new A();  a2.start();  A a3 = new A();  a3.start();      B b1 = new B();  Thread t1 = new Thread(b1);  t1.start();    Thread t2 = new Thread(b1);  t2.start();    Thread t3 = new Thread(b1);  t3.start();                }  }  // check whether it compiles sucessful or not |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | | |
| **Correct Answer: A** | | | |

|  |  |
| --- | --- |
| **171.** | package com.lara.pack1;  public class M1 {  public static void main(String[] args) {  Thread.State[] states = Thread.State.values();  for(Thread.State state : states)  {  System.out.println(state);  }  }  } |
| |  | | --- | | A.  Compile Time error |  |  | | --- | | B.  Runtime Error |  |  | | --- | | C.  NEW  RUNNABLE  BLOCKED  WAITING  TIMED\_WAITING  TERMINATED |  |  | | --- | | D.  NEW  RUNNABLE  BLOCKED  TERMINATED | | | |
| **Correct Answer: C** | | |
| **172.** | package com.lara.pack1;  class A extends Thread  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(i);  }  }  }  public class M2 {  public static void main(String[] args) throws InterruptedException{  A a1 = new A();  System.out.println("a:" + a1.getState());  a1.start();  System.out.println("b:" + a1.getState());  Thread.sleep(7);  System.out.println("c:" + a1.getState());  Thread.sleep(10000);  System.out.println("d:" + a1.getState());  }  } |
| |  | | --- | | A.  SequentialOutput with thread states |  |  | | --- | | B.  Simultaneous Output with thread states |  |  | | --- | | C.  Infinate Output with thread states |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |

|  |  |
| --- | --- |
| **173.** | package com.lara.pack1;  class B extends Thread  {  @Override  public void run() {  try  {  sleep(10000);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  }  public class M3 {  public static void main(String[] args) throws InterruptedException{  B b1 = new B();  b1.start();  Thread.sleep(2000);  System.out.println(b1.getState());  }  } |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  BLOCKED |  |  | | --- | | C.  TIMED\_WAITING |  |  | | --- | | D.  RUNNABLE | | | |
| **Correct Answer: C** | | |
| **174.** | package com.lara.pack1;  class C extends Thread{  @Override  public void run() {  synchronized (this) {  try  {  wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  }  }  public class M4 {  public static void main(String[] args) throws InterruptedException {  C c1 = new C();  c1.start();  Thread.sleep(100);  System.out.println(c1.getState());  }  } |
| |  | | --- | | A.  WAITING |  |  | | --- | | B.  RUNNING |  |  | | --- | | C.  RUNNABLE |  |  | | --- | | D.  TERMINATING | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **175.** | package com.lara.pack1;  class D extends Thread {    Thread mainThread;    D(Thread mainThread)  {  this.mainThread = mainThread;  }    @Override  public void run() {  try  {  sleep(100);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("state of main thread:" + mainThread.getState());  }  }  public class M5 {  public static void main(String[] args) {  Thread t1 = Thread.currentThread();  D d1 = new D(t1);  d1.start();  try  {  d1.join();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }    }  } | |
| |  | | --- | | A.  state of main thread:RUNNING |  |  | | --- | | B.  state of main thread:WAITING |  |  | | --- | | C.  state of main thread:TERMINATING |  |  | | --- | | D.  Compilation Error |  |  | | --- | | E.  Runtime Error | | | | |
| **Correct Answer: B** | | | |
| **176.** | package com.lara.pack1;  class E extends Thread  {  @Override  public void run() {  for(int i = 1; i <= 1000; i++)  {  System.out.println(i);  yield();  }  }  }  class F extends Thread  {  @Override  public void run() {  for(int i = 2000; i <= 3000; i++)  {  System.out.println(i);  yield();  }  }  }  public class M6 {  public static void main(String[] args) {  E e1 = new E();  e1.start();    F f1 = new F();  f1.start();  }  } |
| |  | | --- | | A.  SequentialOutput |  |  | | --- | | B.  Simultaneous Output |  |  | | --- | | C.  Infinate Output |  |  | | --- | | D.  Compilation Error | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **177.** | package com.lara.pack1;  class Shared  {  synchronized void test1()  {  Thread t1 = Thread.currentThread();  for(int i = 1; i <= 1000; i++)  {  System.out.println("from test1 " + i + " by " + t1.getName());  }  }  synchronized void test2()  {  Thread t1 = Thread.currentThread();  for(int i = 1; i <= 1000; i++)  {  System.out.println("from test2 " + i + " by " + t1.getName());  }  }  }  class Thread1 extends Thread  {  Shared s1;  Thread1(Shared s1)  {  this.s1 = s1;  }  @Override  public void run() {  s1.test1();  }  }  class Thread2 extends Thread  {  Shared s1;  Thread2(Shared s1)  {  this.s1 = s1;  }  @Override  public void run() {  s1.test1();  }  }  public class M1 {  public static void main(String[] args) {  Shared s1 = new Shared();  Shared s2 = new Shared();    Thread1 t1 = new Thread1(s1);  Thread2 t2 = new Thread2(s2);    t1.start();  t2.start();  }  } |
| |  | | --- | | A.  Sequential output |  |  | | --- | | B.  simultaneous output |  |  | | --- | | C.  No output |  |  | | --- | | D.  Compilation Error | | | | |
| **Correct Answer: B** | | | |
| **178.** | package com.lara.pack2;  class Shared  {  void test1()  {  Thread t1 = Thread.currentThread();  synchronized(this)  {  for(int i = 1; i <= 1000; i++)  {  System.out.println("from test1 " + i + " by " + t1.getName());  }  }  }  void test2()  {  Thread t1 = Thread.currentThread();  synchronized(this)  {  for(int i = 1; i <= 1000; i++)  {  System.out.println("from test2 " + i + " by " + t1.getName());  }  }  }  }  class Thread1 extends Thread  {  Shared s1;  Thread1(Shared s1)  {  this.s1 = s1;  }  @Override  public void run() {  s1.test1();  }  }  class Thread2 extends Thread  {  Shared s1;  Thread2(Shared s1)  {  this.s1 = s1;  }  @Override  public void run() {  s1.test2();  }  }  public class M1 {  public static void main(String[] args) {  Shared s1 = new Shared();  Shared s2 = new Shared();    Thread1 t1 = new Thread1(s1);  Thread2 t2 = new Thread2(s2);    t1.start();  t2.start();  }  } | |
| |  | | --- | | A.  Sequential output |  |  | | --- | | B.  simultaneous output |  |  | | --- | | C.  No output |  |  | | --- | | D.  Compilation Error | | | | |
| **Correct Answer: B** | | | |

|  |  |
| --- | --- |
| **179.** | package com.lara.pack3;  class A  {  static void test1()  {  Thread t1 = Thread.currentThread();  synchronized (A.class)  {  for(int i = 1; i <= 1000; i++)  {  System.out.println("from test1 with " + i + " by " + t1.getName());  }  }  }  static void test2()  {  Thread t1 = Thread.currentThread();  synchronized (A.class)  {  for(int i = 1; i <= 1000; i++)  {  System.out.println("from test2 with " + i + " by " + t1.getName());  }  }  }  }  class Thread1 extends Thread  {  @Override  public void run() {  A.test1();  }  }  class Thread2 extends Thread  {  @Override  public void run() {  A.test2();  }  }  public class M1 {  public static void main(String[] args) {  Thread1 t1 = new Thread1();  t1.start();    Thread2 t2 = new Thread2();  t2.start();  }  } |
| |  | | --- | | A.  Sequential output |  |  | | --- | | B.  simultaneous output |  |  | | --- | | C.  No output |  |  | | --- | | D.  Compilation Error | | | |
| **Correct Answer: A** | | |
| **180.** | package com.lara.pack4;  import java.lang.management.ManagementFactory;  import java.lang.management.ThreadMXBean;  import java.util.Arrays;  class Shared  {  synchronized void test1(Shared obj)  {  Thread t1 = Thread.currentThread();  System.out.println("test1 begin by " + t1.getName());  Util.sleep(1000);  obj.test2(this);  System.out.println("test1 end by " + t1.getName());  }  synchronized void test2(Shared obj)  {  Thread t1 = Thread.currentThread();  System.out.println("test2 begin by " + t1.getName());  Util.sleep(1000);  obj.test1(this);  System.out.println("test2 end by " + t1.getName());  }  }  class Util  {  static void sleep(long millis)  {  try  {  Thread.sleep(millis);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  }  class Thread1 extends Thread  {  Shared s1, s2;  Thread1(Shared s1, Shared s2)  {  this.s1 = s1;  this.s2 = s2;  }  @Override  public void run() {  s1.test1(s2);  }  }  class Thread2 extends Thread  {  Shared s1, s2;  Thread2(Shared s1, Shared s2)  {  this.s1 = s1;  this.s2 = s2;  }  @Override  public void run() {  s2.test2(s1);  }  }  public class M1 {  public static void main(String[] args) {  Shared s1 = new Shared();  Shared s2 = new Shared();  Thread1 t1 = new Thread1(s1, s2);  t1.start();  Util.sleep(100);  Thread2 t2 = new Thread2(s1, s2);  t2.start();    Util.sleep(2000);    ThreadMXBean tmx = ManagementFactory.getThreadMXBean();  long[] ids = tmx.findDeadlockedThreads();    if(ids != null)  {  System.out.println("Threads are under dead lock");  System.out.println("dead locked thread ids: " + Arrays.toString(ids));  }  else  {  System.out.println("no threads are under dead lock");  }  System.out.println(t1.getState());  System.out.println(t2.getState());    }  } |
| |  | | --- | | A.  Sequential output |  |  | | --- | | B.  simultaneous output |  |  | | --- | | C.  No output |  |  | | --- | | D.  test1 begin by Thread-0  test2 begin by Thread-1  Threads are under dead lock  dead locked thread ids: [11, 10]  BLOCKED  BLOCKED |  |  | | --- | | E.  Compilation error | | | |
| **Correct Answer: D** | | |

|  |  |
| --- | --- |
| **181.** | package com.lara.pack5;  class A  {  synchronized void test1()  {  System.out.println("test1 on " + this + " begin by " + Thread.currentThread().getName());  try  {  wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("test1 on " + this + " end by " + Thread.currentThread().getName());  }  synchronized void test2()  {  System.out.println("test2 on " + this + " begin by " + Thread.currentThread().getName());  notifyAll();  System.out.println("test2 on " + this + " end by " + Thread.currentThread().getName());  }  }  class Thread1 extends Thread  {  A obj;  Thread1(A obj)  {  this.obj = obj;  }  @Override  public void run() {  obj.test1();  }  }  class Thread2 extends Thread  {  A obj;  Thread2(A obj)  {  this.obj = obj;  }  @Override  public void run() {  obj.test1();  }  }  public class M1 {  public static void main(String[] args) throws InterruptedException{  A obj1 = new A();  A obj2 = new A();  Thread1 t1 = new Thread1(obj1);  t1.start();  Thread2 t2 = new Thread2(obj1);  t2.start();  Thread.sleep(10000);  System.out.println("main after 10 sec sleep");  obj2.test2();  }  }  //Check whether this program threads under dead lock or not |
| |  | | --- | | A.  Yes |  |  | | --- | | B.  No | | | | |
| **Correct Answer: B** | | | |
| **182.** | package com.lara.pack6;  class A  {  synchronized void test1()  {  System.out.println("test1 on " + this + " begin by " + Thread.currentThread().getName());  try  {  wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("test1 on " + this + " end by " + Thread.currentThread().getName());  }  }  class Thread1 extends Thread  {  A obj;  Thread1(A obj)  {  this.obj = obj;  }  @Override  public void run() {  obj.test1();  }  }  class Thread2 extends Thread  {  A obj;  Thread2(A obj)  {  this.obj = obj;  }  @Override  public void run() {  obj.test1();  }  }  public class M1 {  public static void main(String[] args) throws InterruptedException{  A obj1 = new A();  A obj2 = new A();  Thread1 t1 = new Thread1(obj1);  t1.start();  Thread2 t2 = new Thread2(obj1);  t2.start();  Thread.sleep(10000);  System.out.println("main after 10 sec sleep");  synchronized (obj1)  {  System.out.println("test2 on " + obj1 + " begin by " + Thread.currentThread().getName());  obj1.notifyAll();  System.out.println("test2 on " + obj1 + " end by " + Thread.currentThread().getName());  }  }  } | |
| |  | | --- | | A.  threads under dead lock |  |  | | --- | | B.  No dead lock |  |  | | --- | | C.  test1 on th.A@2d4021ce begin by Thread-0  test1 on th.A@2d4021ce begin by Thread-1  main after 10 sec sleep  test2 on th.A@2d4021ce begin by main  test2 on th.A@2d4021ce end by main  test1 on th.A@2d4021ce end by Thread-1  test1 on th.A@2d4021ce end by Thread-0 |  |  | | --- | | D.  test1 on th.A@2d4021ce begin by Thread-0  test1 on th.A@2d4021ce begin by Thread-1  main after 10 sec sleep  test2 on th.A@2d4021ce begin by main  test2 on th.A@2d4021ce end by main  test1 on th.A@2d4021ce end by Thread-0  test1 on th.A@2d4021ce end by Thread-1 | | | | |
| **Correct Answer: C** | | | |

|  |  |
| --- | --- |
| **183.** | package com.lara.pack7;  class A  {  }  class Thread1 extends Thread  {  A obj;  Thread1(A obj)  {  this.obj = obj;  }  @Override  public void run() {    synchronized (obj)  {  System.out.println("wait on " + obj + " begin by " + Thread.currentThread().getName());  try  {  obj.wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("wait on " + obj + " end by " + Thread.currentThread().getName());  }  }  }  class Thread2 extends Thread  {  A obj;  Thread2(A obj)  {  this.obj = obj;  }  @Override  public void run() {  synchronized (obj)  {  System.out.println("wait on " + obj + " begin by " + Thread.currentThread().getName());  try  {  obj.wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("wait on " + obj + " end by " + Thread.currentThread().getName());  }  }  }  public class M1 {  public static void main(String[] args) throws InterruptedException{  A obj1 = new A();  A obj2 = new A();  Thread1 t1 = new Thread1(obj1);  t1.start();  Thread2 t2 = new Thread2(obj1);  t2.start();  Thread.sleep(10000);  System.out.println("main after 10 sec sleep");  synchronized (obj1)  {  System.out.println("test2 on " + obj1 + " begin by " + Thread.currentThread().getName());  obj1.notifyAll();  System.out.println("test2 on " + obj1 + " end by " + Thread.currentThread().getName());  }  }  } |
| |  | | --- | | A.  threads under waiting |  |  | | --- | | B.  No waiting |  |  | | --- | | C.  wait on th.A@64fe8248 begin by Thread-0  wait on th.A@64fe8248 begin by Thread-1  main after 10 sec sleep  test2 on th.A@64fe8248 begin by main  test2 on th.A@64fe8248 end by main  wait on th.A@64fe8248 end by Thread-0  wait on th.A@64fe8248 end by Thread-1 |  |  | | --- | | D.  wait on th.A@64fe8248 begin by Thread-0  wait on th.A@64fe8248 begin by Thread-1  main after 10 sec sleep  test2 on th.A@64fe8248 begin by main  test2 on th.A@64fe8248 end by main  wait on th.A@64fe8248 end by Thread-1  wait on th.A@64fe8248 end by Thread-0 | | | |
| **Correct Answer: D** | | |
| **184.** | package com.lara.pack8;  class Thread1 extends Thread  {  @Override  public void run() {    synchronized (this)  {  System.out.println("wait on " + this + " begin by " + Thread.currentThread().getName());  try  {  this.wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("wait on " + this + " end by " + Thread.currentThread().getName());  }  }  }  class Thread2 extends Thread  {  public void run() {    synchronized (this)  {  System.out.println("wait on " + this + " begin by " + Thread.currentThread().getName());  try  {  this.wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("wait on " + this + " end by " + Thread.currentThread().getName());  }  }  }  public class M1 {  public static void main(String[] args) throws InterruptedException{  Thread1 t1 = new Thread1();  t1.start();  Thread2 t2 = new Thread2();  t2.start();  Thread.sleep(10000);  System.out.println("main after 10 sec sleep");  synchronized (t1)  {  System.out.println("test2 on " + t1 + " begin by " + Thread.currentThread().getName());  t1.notify();  System.out.println("test2 on " + t1 + " end by " + Thread.currentThread().getName());  }  synchronized (t2)  {  System.out.println("test2 on " + t2 + " begin by " + Thread.currentThread().getName());  t2.notify();  System.out.println("test2 on " + t2 + " end by " + Thread.currentThread().getName());  }      }  } |
| |  | | --- | | A.  wait on Thread[Thread-0,5,main] begin by Thread-0  wait on Thread[Thread-1,5,main] begin by Thread-1  main after 10 sec sleep  test2 on Thread[Thread-0,5,main] begin by main  test2 on Thread[Thread-0,5,main] end by main  test2 on Thread[Thread-1,5,main] begin by main  wait on Thread[Thread-0,5,main] end by Thread-0  test2 on Thread[Thread-1,5,main] end by main  wait on Thread[Thread-1,5,main] end by Thread-1 |  |  | | --- | | B.  wait on Thread[Thread-0,5,main] begin by Thread-1  wait on Thread[Thread-1,5,main] begin by Thread-0  main after 10 sec sleep  test2 on Thread[Thread-0,5,main] begin by main  test2 on Thread[Thread-0,5,main] end by main  test2 on Thread[Thread-1,5,main] begin by main  wait on Thread[Thread-0,5,main] end by Thread-1  test2 on Thread[Thread-1,5,main] end by main  wait on Thread[Thread-1,5,main] end by Thread-0 |  |  | | --- | | C.  wait on Thread[Thread-0,5,main] begin by Thread-0  wait on Thread[Thread-1,5,main] begin by Thread-1  main after 10 sec sleep  wait on Thread[Thread-0,5,main] end by Thread-0  test2 on Thread[Thread-1,5,main] end by main  wait on Thread[Thread-1,5,main] end by Thread-1 |  |  | | --- | | D.  None | | | |
| **Correct Answer: A** | | |

|  |  |  |
| --- | --- | --- |
| **185.** | package com.lara.pack9;  class Thread1 extends Thread  {  @Override  public void run() {    synchronized (this)  {  System.out.println("wait on " + this + " begin by " + Thread.currentThread().getName());  try  {  this.wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  System.out.println("wait on " + this + " end by " + Thread.currentThread().getName());  }  }  }  class Thread2 extends Thread  {  Thread1 obj;  Thread2(Thread1 obj)  {  this.obj = obj;  }  public void run() {    synchronized (obj)  {  System.out.println("notify on " + obj + " begin by " + Thread.currentThread().getName());  obj.notify();  System.out.println("notify on " + obj + " end by " + Thread.currentThread().getName());  }  }  }  public class M1 {  public static void main(String[] args) throws InterruptedException{  Thread1 t1 = new Thread1();  t1.start();  Thread.sleep(10000);  System.out.println("main after 10 sec sleep");  Thread2 t2 = new Thread2(t1);  t2.start();  }  } | |
| |  | | --- | | A.  wait on Thread[Thread-0,5,main] begin by Thread-1  main after 10 sec sleep  notify on Thread[Thread-0,5,main] begin by Thread-0  notify on Thread[Thread-0,5,main] end by Thread-0  wait on Thread[Thread-0,5,main] end by Thread-1 |  |  | | --- | | B.  wait on Thread[Thread-0,5,main] begin by Thread-0  main after 10 sec sleep  notify on Thread[Thread-0,5,main] begin by Thread-1  notify on Thread[Thread-0,5,main] end by Thread-1  wait on Thread[Thread-0,5,main] end by Thread-0 |  |  | | --- | | C.  Compilation Error |  |  | | --- | | D.  None | | | | |
| **Correct Answer: B** | | | |
| **186.** | package com.lara.pack10;  import java.util.ArrayList;  class Util  {  static void sleep(long millis)  {  try  {  Thread.sleep(millis);  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }  }  class CommonTaskThread extends Thread  {  @Override  public void run()  {  while(true)  {  //wait  synchronized (this) {  try  {  wait();  }  catch(InterruptedException ex)  {  ex.printStackTrace();  }  }    //common task. it can be any  for(int i = 1; i <= 10; i++)  {  System.out.println(i + " by " + getName());  Util.sleep(1000);  }    //notify  synchronized (this) {  notify();  }  }  }  }  class ThreadPoolManager  {  private ArrayList<CommonTaskThread> pool = new ArrayList<CommonTaskThread>();    public void init()  {  CommonTaskThread ct = null;  for(int i = 1; i <= 10; i++)  {  ct = new CommonTaskThread();  ct.start();  pool.add(ct);  }  }    public CommonTaskThread checkOut()  {  CommonTaskThread ct = null;  if(pool.size() > 0)  {  ct = pool.remove(0);  }  else  {  ct = new CommonTaskThread();  ct.start();  }  return ct;  }    public void checkIn(CommonTaskThread ct)  {  if(pool.size() < 10)  {  pool.add(ct);  }  else  {  ct = null;  }  }    public void release()  {  CommonTaskThread ct = null;  for(int i = 0; i < pool.size(); )  {  ct = pool.remove(0);  ct = null;  }  pool = null;  }  }  class Customer1 extends Thread  {  private ThreadPoolManager tpm;  Customer1(ThreadPoolManager tpm)  {  this.tpm = tpm;  }    @Override  public void run() {  while(true)  {  System.out.println("customer1 is trying to get a thread from the pool");  CommonTaskThread ct = tpm.checkOut();  synchronized (ct) {  ct.notify();  }  synchronized (ct) {  try {  ct.wait();  }  catch(InterruptedException ex) {  ex.printStackTrace();  }  }  System.out.println("customer1 is sending used thread back to the pool");  tpm.checkIn(ct);  Util.sleep(10000);  }  }  }  class Customer2 extends Thread  {  private ThreadPoolManager tpm;  Customer2(ThreadPoolManager tpm)  {  this.tpm = tpm;  }    @Override  public void run() {  while(true)  {  System.out.println("customer2 is trying to get a thread from the pool");  CommonTaskThread ct = tpm.checkOut();  synchronized (ct) {  ct.notify();  }  synchronized (ct) {  try {  ct.wait();  }  catch(InterruptedException ex) {  ex.printStackTrace();  }  }  System.out.println("customer2 is sending used thread back to the pool");  tpm.checkIn(ct);  Util.sleep(10000);  }  }  }  class Customer3 extends Thread  {  private ThreadPoolManager tpm;  Customer3(ThreadPoolManager tpm)  {  this.tpm = tpm;  }    @Override  public void run() {  while(true)  {  System.out.println("customer3 is trying to get a thread from the pool");  CommonTaskThread ct = tpm.checkOut();  synchronized (ct) {  ct.notify();  }  synchronized (ct) {  try {  ct.wait();  }  catch(InterruptedException ex) {  ex.printStackTrace();  }  }  System.out.println("customer3 is sending used thread back to the pool");  tpm.checkIn(ct);  Util.sleep(10000);  }  }  }  public class M1 {  public static void main(String[] args) {  ThreadPoolManager tpm = new ThreadPoolManager();  tpm.init();    Customer1 c1 = new Customer1(tpm);  c1.start();    Customer2 c2 = new Customer2(tpm);  c2.start();    Customer3 c3 = new Customer3(tpm);  c3.start();    Util.sleep(1000 \* 1000);  c1.stop();  c2.stop();  c3.stop();  Util.sleep(20000);  tpm.release();  System.out.println("end of the game");  }  } |
| |  | | --- | | A.  Sequential output |  |  | | --- | | B.  Simultaneous output |  |  | | --- | | C.  infinite output |  |  | | --- | | D.  None | | | | |
| **Correct Answer: C** | | | |

Bottom of Form