01. [Sebesta, 2000] Quais são os três níveis de concorrência nos programas?

02. [Sebesta, 2000] Qual nível de concorrência de programa é melhor suportado por computadores SIMD?

03. [Sebesta, 2000] Qual nível de concorrência de programa é melhor suportado por computadores MIMD?

04. [Sebesta, 2000] O que é o *thread* de controle de um programa?

05. [Sebesta, 2000] Defina tarefa, tarefa disjunta, sincronização, sincronização de competição e cooperação, vivência e *deadlock*.

06. [Sebesta, 2000] Quais tipos de tarefa não exigem nenhum tipo de sincronização?

07. [Sebesta, 2000] Quais são as questões de projeto referentes ao suporte de linguagem para concorrência?

08. [Sebesta, 2000] Descreva as ações das operações de espera (*wait*) e de liberação (*release*) para semáforos.

09. [Sebesta, 2000] O que é um semáforo binário? O que é um de contagem?

10. [Sebesta, 2000] Quais são os principais problemas decorrentes do uso de semáforos para fornecer sincronização?

11. [Sebesta, 2000] Qual vantagem os monitores têm sobre os semáforos?

12. [Sebesta, 2000] Defina *rendezvous*, cláusula `accept`, cláusula `entry`, tarefa atuante, tarefa servidora, cláusula `accept` estendida, cláusula `accept` aberta, cláusula `accept` fechada e tarefa concluída.

13. [Sebesta, 2000] Qual é mais geral: a concorrência por monitores ou a concorrência por passagem de mensagens?

14. [Sebesta, 2000] As tarefas Ada são criadas estática ou dinamicamente?

15. [Sebesta, 2000] Para que propósito serve uma cláusula `accept` estendida?

16. [Sebesta, 2000] Como a sincronização de cooperação é fornecida para tarefas Ada?

17. [Sebesta, 2000] Qual é a vantagem dos objetos protegidos da Ada 95 para oferecer acesso a objetos dados compartilhados?

18. [Sebesta, 2000] Descreva a cláusula `select` assíncrona da Ada 95.

19. [Sebesta, 2000] Especificamente, qual unidade de programa Java pode rodar concorrentemente com o método principal em um programa aplicativo?

20. [Sebesta, 2000] O que o método `sleep` Java faz?

21. [Sebesta, 2000] O que o método `yield` Java faz?

22. [Sebesta, 2000] Quais são as duas construções Java que podem ser sincronizadas?

23. [Sebesta, 2000] Quais métodos Java são usados para suportar sincronização de cooperação?

24. [Sebesta, 2000] Explique porque o Java inclui a interface `Runnable`.

25. [Sebesta, 2000] Qual é o objetivo dos comandos de especificação do *High-Performance* FORTRAN?

26. [Sebesta, 2000] Qual a finalidade do comando `FORALL` do *High-Performance* FORTRAN?

27. [Sebesta, 2000] Explique claramente por que a sincronização de competição não é um problema em um ambiente de programação que tem controle de unidade simétrico, mas sem concorrência.

28. [Sebesta, 2000] Qual é a melhor ação que um sistema pode desenvolver quando um *deadlock* é detectado?

29. [Sebesta, 2000] Escreva uma tarefa Ada para implementar semáforos gerais.

30. [Sebesta, 2000] Escreva uma tarefa Ada para gerenciar um *buffer* compartilhado como o de nosso exemplo, mas use a tarefa de semáforo do Problema 29.

31. [Sebesta, 2000] Espera ocupada (*busy waiting*) é um método por meio do qual uma tarefa espera que determinado evento verifique continuamente se esse evento ocorre. Qual é o problema com essa abordagem?

32. [Sebesta, 2000] No exemplo do produtor-consumidor da Seção 12.3, suponhamos que substituímos incorretamente a `release(access)` do processo consumidor por `wait(access)`. Qual seria o resultado desse erro na execução do sistema?

33. [Sebesta, 2000] De um livro sobre programação em linguagem *assembly* VAX, determine quais instruções a estrutura VAX inclui para suportar a construção de semáforos.

34. [Sebesta, 2000] De um livro sobre programação em linguagem *assembly* para um computador que usa um processador Intel Pentium, determine quais instruções são fornecidas para suportar a construção de semáforos.

35. [Sebesta, 2000] Suponhamos que duas tarefas, `A` e `B`, devam usar a variável compartilhada `TAM_BUF`. A tarefa `A` adiciona `2` a `TAM_BUF`, e a tarefa `B` subtrai `1` dela. Suponhamos que tais operações aritméticas sejam feitas pelo processo em três etapas: buscar o valor atual, realizar a operação aritmética e colocar de volta o novo valor. Na ausência de sincronização de competição, quais sequências de eventos são possíveis e quais valores resultam dessas operações? Suponhamos que o valor inicial de `TAM_BUF` seja `6`.

36. [Sebesta, 2000] Compare o mecanismo de sincronização de competição do Java com o da Ada.

37. [Sebesta, 2000] Compare o mecanismo de sincronização de cooperação do Java com o da Ada.

38. [Sebesta, 2000] O que acontece se um procedimento monitor chamar outro procedimento no mesmo monitor?

39. [Sierra, 2004] Dado o código a seguir,

```
1.  class MyThread extends Thread {
2.
3.  public static void main(String[] args) {
4.  MyThread t = new MyThread();
5.  t.run();
6.  }
7.
8.  public void run() {
9.  for(int i=1;i<3;++i) {
10. System.out.println(i + "..");
11. }
12. }
13. }
```

Qual será o resultado?

a) Esse código não será compilado devido à linha 4.

b) Esse código não será compilado devido à linha 5.

c) `1..2..`

d) `1..2..3..`

e) Uma exceção será lançada no tempo de execução.

40. [Sierra, 2004] Dois dos métodos abaixo são definidos na classe `Thread`. Quais?

a) `start()`

b) `wait()`

c) `notify()`

d) `run()`

e) `terminate()`

41. [Sierra, 2004] O bloco de código a seguir cria um objeto `Thread` que usa um destino `Runnable`:

```
Runnable target = new Runnable();
Thread myThread = new Thread(runnable);
```

Qual das classes abaixo pode ser usada na criação do destino, para que o código anterior seja compilado corretamente?

a) `public class MyRunnable extends Runnable {public void run(){}}`

b) `public class MyRunnable extends Object {public void run(){}}`

c) `public class MyRunnable implements Runnable {public void run(){}}`

d) `public class MyRunnable implements Runnable {void run(){}}`

e) `public class MyRunnable implements Runnable {pubic void start(){}}`

42. [Sierra, 2004] Dado o código a seguir,

```
1.  class MyThread extends Thread {
2.
3.  public static void main(String[] args) {
4.  MyThread t = new MyThread();
5.  t.start();
6.  System.out.println("one. ");
7.  t.start();
8.  System.out.println("two. ");
9.  }
10.
11. public void run() {
12. System.out.println("Thread ");
13. }
14. }
```

Qual será o resultado?

a) a compilação falhará

b) uma exceção ocorrerá no tempo de execução

c) `Thread one. Thread two.`

d) A saída não pode ser determinada.

43. [Sierra, 2004] Dado o código a seguir,

```
1. public class MyRunnable implements Runnable {
2. public void run() {
3. // some code here
4. }
5. }
```

Qual dessas opções criará e iniciará esse segmento?

a) `new Runnable(MyRunnable).start();`

b) `new Thread(MyRunnable).start();`

c) `new Thread(new MyRunnable()).start();`

d) `new MyRunnable().start();`

44. [Sierra, 2004] Dado o código a seguir,

```
1.  class MyThread extends Thread {
2.
3.  public static void main(String[] args) {
4.  MyThread t = new MyThread();
5.  Thread x = new Thread(t);
6.  x.start();
7.  }
8.
9.  public void run() {
10. for(int i=0;i<3;++i) {
11. System.out.println(i + "..");
12. }
13. }
```

Qual será seu resultado?

a) a compilação falhará

b) `1..2..3..`

c) `0..1..2..3..`

d) `0..1..2..`

e) uma exceção ocorrerá no tempo de execução.

45. [Sierra, 2004] Dado o código a seguir,

```
1.  class Test {
2.
3.  public static void main(String[] args) {
4.  printAll(args);
5.  }
6.
7.  public static void printAll(String[] lines) {
8.  for(int i=0;i<lines.length;i++) {
9.  System.out.println(lines[i]);
10. Thread.currentThread().sleep(1000);
11. }
12. }
```

```
13. }
```

O método estático `Thread.currentThread()` retornará uma referência ao objeto `Thread` que estiver sendo executado no momento. Qual será o resultado desse código?

a) Será exibida cada `String` das linhas do *array*, com uma pausa de 1 segundo.

b) Será exibida cada `String` das linhas do *array*, sem pausa entre elas porque esse método não é executado em um objeto `Thread`.

c) Será exibida cada `String` das linhas do *array*, sem garantias de que haverá uma pausa porque `currentThread()` pode não recuperar esse segmento.

d) Esse código não será compilado.

46. [Sierra, 2004] Suponhamos que você tivesse uma classe com duas variáveis `private`: `a` e `b`. Quais dos pares a seguir podem impedir problemas de acesso simultâneo nessa classe (selecione todos que forem aplicáveis)?

a)
```
public int read(int a, int b) {return a + b;}
public void set(int a, int b) {this.a=a;thisb=b;}
```

b)
```
public synchronized int read(int a, int b) {return a + b;}
public synchronized void set(int a, int b) {this.a=a;thisb=b;}
```

c)
```
public int read(int a, int b) {synchronized(a) {return a + b;}}
public void set(int a, int b) {synchronized(a) {this.a=a;thisb=b;}}
```

d)
```
public int read(int a, int b) {synchronized(a) {return a + b;}}
public void set(int a, int b) {synchronized(b) {this.a=a;thisb=b;}}
```

e)
```
public synchronized(this) int read(int a, int b) {return a + b;}
public synchronized(this) void set(int a, int b) {this.a=a;thisb=b;}
```

f)
```
public int read(int a, int b) {synchronized(this) {return a + b;}}
public void set(int a, int b) {synchronized(this) {this.a=a;thisb=b;}}
```

47. [Sierra, 2004] Que classe ou interface define os métodos `wait()`, `notify()` e `notifyAll()`?

a) `Object`

b) `Thread`

c) `Runnable`

d) `Class`

48. [Sierra, 2004] Duas declarações são verdadeiras. Quais?

a) Um método estático não pode ser sincronizado.

b) Mesmo se uma classe tiver um código sincronizado, vários segmentos poderão acessar o código não-incronizado.

c) As variáveis podem ser protegidas de problemas de acesso simultâneo sendo marcadas com a palavra-have `synchronized`.

d) Quando um segmento entra em suspensão, libera seus bloqueios.

e) Quando um segmento chama `wait()`, libera seus bloqueios.

49. [Sierra, 2004] Três dos métodos abaixo são da classe `Object`. Quais (selecione três)?

a) `notify();`

b) `notifyAll();`

c) `isInterrupted();`

d) `synchronized();`

e) `interrupt();`

f) `wait(long msecs);`

g) `sleep(long msecs);`

h) `yield();`

50. [Sierra, 2004] Dado o código a seguir,

```
1.  public class WaitTest {
2.  public static void main(String[] args) {
3.  System.out.println("1 ");
4.  synchronized(args) {
5.  System.out.println("2 ");
6.  try {
7.  args.wait();
8.  }
9.  catch(InterruptedException e) {}
10. }
11. System.out.println("3 ");
12. }
13. }
```

Qual será o resultado se tentarmos compilar e executar esse programa?

a) sua compilação falhará porque a exceção `IllegalMonitorStateException` de `wait()` não é manipulada na linha 7.

b) `1 2 3`

c) `1 3`

d) `1 2`

e) no tempo de execução, ele lançará uma exceção `IllegalMonitorStateException` quando tentar entrar no estado de espera.

f) sua compilação falhará porque ele tem que ser sincronizado no objeto `this`.

51. [Sierra, 2004] Suponhamos que o método a seguir tenha sido apropriadamente sincronizado e chamado por um segmento A no objeto B: `wait(2000);`

Depois que esse método for chamado, quando o segmento A se tornará candidato a conseguir outra chance de ser executado pela CPU?

a) depois que o segmento A for notificado, ou após dois segundos.

b) depois que o bloqueio de B for liberado, ou após dois segundos.

c) dois segundos após o segmento A ser notificado.

d) dois segundos após o bloqueio de B ser liberado.

52. [Sierra, 2004] Duas declarações são verdadeiras. Quais?

a) o método `notifyAll()` deve ser chamado em um contexto sincronizado.

b) para chamar `wait()`, um objeto deve possuir o bloqueio do segmento.

c) o método `notify()` é definido na classe `java.lang.Thread`

d) quando um segmento estiver aguardando como resultado de `wait()`, liberará seus bloqueios.

e) o método `notify()` faz com que um segmento libere imediatamente seus bloqueios.

f) a diferença entre `notify()` e `notifyAll()` é que `notifyAll()` notifica todos os segmentos que estão aguardando, independente do objeto em que estão esperando.

53. [Sierra, 2004] Suponhamos que você criasse um programa e um de seus segmentos (chamado `backgroundThread`) executasse um processamento numérico extenso. Qual seria a maneira apropriada de configurar sua prioridade na tentativa de fazer com que o resto do sistema continuasse com boa capacidade de resposta enquanto o segmento fosse processado? (Selecione todas que forem aplicáveis).

a) `backgroundThread.setPriority(Thread.LOW_PRIORITY);`

b) `backgroundThread.setPriority(Thread.MAX_PRIORITY);`

c) `backgroundThread.setPriority(1);`

d) `backgroundThread.setPriority(Thread.NO_PRIORITY);`

e) `backgroundThread.setPriority(Thread.MIN_PRIORITY);`

f) `backgroundThread.setPriority(Thread.NORM_PRIORITY);`

g) `backgroundThread.setPriority(10);`

54. [Sierra, 2004] Três das opções abaixo podem garantir que um segmento sairá do estado de execução. Quais?

a) `yield()`

b) `wait()`

c) `notify()`

d) `notifyAll`

e) `sleep(1000)`

f) `aLiveThread.join()`

g) `Thread.killThread()`

55. [Sierra, 2004] Duas declarações são verdadeiras. Quais?

a) o impasse não ocorrerá se `wait()`/`notify()` forem usados.

b) um segmento começará a ser executado novamente logo que a duração de sua suspensão expirar.

c) a sincronização pode impedir que dois objetos sejam acessados pelo mesmo segmento.

d) o método `wait()` é sobreposto para aceitar um período de tempo.

e) o método `notify()` é sobreposto para aceitar um período de tempo.

f) tanto `wait()` quanto `notify()` devem ser chamados em um contexto sincronizado.

g) `wait()` não lançará uma exceção verificada.

h) `sleep()` pode lançar uma exceção de tempo de execução.

56. [Sierra, 2004] Duas das opções abaixo são construtores válidos para `Thread`. Quais?

a) `Thread(Runnable r, String name)`

b) `Thread()`

c) `Thread(int priority)`

d) `Thread(Runnable r, ThreadGroup g)`

e) `Thread(Runnable r, int priority)`

57. [Sierra, 2004] Dado o código a seguir,

```
class MyThread extends Thread {
MyThread() {
System.out.println(" MyThread");
}
public void run() {
System.out.println(" bar");
}
public void run(String s) {
System.out.println(" baz");
}
}
public class TestThreads {
public static void main(String[] args) {
Thread t = new MyThread() {
public void run() {
System.out.println(" foo");
}
};
t.start();
}
}
```

Qual será o resultado?

a) `foo`

b) `MyThread foo`

c) `MyThread bar`

d) `foo bar`

e) `foo bar baz`

f) `bar foo`

g) a compilação falhará

58. [Sierra, 2004] Dado o código a seguir,

```
public class SyncTest {
public static void main(String[] args) {
Thread t = new Thread() {
Foo f = new Foo();
public void run() {
f.increase(20);
}
};
t.start();
}
}
class Foo {
private int data = 23;
public void increase(int amt) {
int x = data;
data = x + amt;
}
}
```

E presumindo que os dados precisem de proteção contra danos, o que – se houver algo – você pode adicionar ao código anterior para assegurar a integridade dos dados?

a) sincronizar o método `run`.

b) inserir `syncronize(this)` na chamada a `f.increase(20)`.

c) o código dado não será compilado.

d) o código dado causará uma exceção de tempo de execução.

e) inserir uma chamada a `wait()` antes da chamada ao método `increase()`.

f) sincronizar o método `increase()`.

59. [Sierra, 2004] Dado o código a seguir,

```
1.  public class Test {
2.  public static void main(String[] args) {
3.  final Foo f = new Foo();
4.  Thread t = new Thread(new Runnable() {
5.  public void run() {
6.  f.doStuff();
7.  }
8.  });
9.  Thread g = new Thread() {
10. public void run() {
11. f.doStuff();
12. }
13. };
14. t.start();
15. g.start();
16. }
17. }
```

```
1.  class Foo {
2.  int x = 5;
3.  public void doStuff() {
4.  if(x < 10) {
5.  // nothing to do
6.  try {
7.  wait();
8.  } catch(InterruptedException ex){}
9.  } else {
10. System.out.println("x is" + x++);
11. if(x >= 10) {
12. notify();
13. }
14. }
15. }
16. }
```

Qual será o resultado?

a) o código não será compilado por causa de um erro na linha 12 da classe `Foo`.

b) o código não será compilado por causa de um erro na linha 7 da classe `Foo`.

c) o código não será compilado por causa de um erro na linha 4 da classe `Test`.

d) o código não será compilado por causa de algum outro erro na classe `Test`.

e) uma exceção ocorrerá no tempo de execução.

f)
```
x is 5
x is 6
```

60. [Sierra, 2004] Nesse exercício criaremos um segmento de contagem simples. Ele contará até 100, fazendo uma pausa de um segundo antes de cada número. Além disso, continuando com o tema da contagem, ele exibirá uma `String` a cada dez números.

Crie uma classe e estenda `Thread`. Como alternativa, você pode implementar a interface `Runnable`.

Substitua o método `run()` de `Thread`. É aí que entrará o código que exibirá os números.

Crie um *loop* `for` que seja executado 100 vezes. Use a operação de resto para verificar a existência de algum número que seja o resto da divisão por 10.

Use o método estático `Thread.sleep()` para fazer uma pausa. Um número do tipo `long` representará os milissegundos.

61. [Sierra, 2004] Neste exercício tentaremos sincronizar um bloco de código. Dentro desse bloco obteremos o bloqueio de um objeto para que outros segmentos não possam alterá-lo enquanto o bloco de código estiver sendo executado. Criaremos três segmentos, todos tentando manipular o mesmo objeto. Cada segmento exibirá a mesma letra 100 vezes e, em seguida, a exibição passará para a próxima letra. Usaremos um objeto `StringBuffer`. Poderíamos executar a sincronização em um objeto `String`, mas as strings não podem ser alteradas depois que são criadas, portanto, não poderíamos passara para a próxima lera sem gerar um novo objeto `String`. A saída final deverá ter 100 As, 100 Bs e 100 Cs, tudo na mesma linha.

Crie a classe e estenda `Thread`.

Substitua o método `run()` de `Thread`. É aí que entrará o bloco de código sincronizado.

Para que nossos três segmentos compartilhem o mesmo objeto, teremos que criar um construtor que

aceite um objeto `StringBuffer` no argumento.

O bloco de código sincronizado obterá um bloqueio do objeto `StringBuffer` declarado na etapa anterior.

Dentro do bloco, exiba o objeto `StringBuffer` 100 vezes e, em seguida, passe para a letra seguinte.

Para concluir, no método `main()`, crie somente um objeto `StringBuffer` que use a letra A e, em seguida, gere três instâncias de nossa classe e inicie todas elas.