Laboratorium 10

CEL: Wątki i synchronizacja.

Zadanie#1. Przeanalizuj poniższy kod.

```
01:
02: class ThreadA implements Runnable {
03:   int k;
04:   public void run() {
05:     for (int i=0; i<3; i++) {
06:       System.out.println("k=" + ++k);
07:     }
08:   }
09: }
10:
11: public class Question{
12:   public static void main(String[] args){
13:     ThreadA a = new ThreadA();
14:     Thread  t = new Thread(a);
15:     t.start();
16:     t.start();
17:   }
18: }
```

Wskaż możliwy wynik/wyniki wykonania powyższego programu.

a)	k=1 k=2 k=3 k=1 k=2 k=3
b)	k=1 k=2 k=3 k=4 k=5 k=6
c)	k=1 k=1 k=2 k=2 k=3 k=3
d)	Runtime error
e)	Compile time error

■

Zadanie#2. Przeanalizuj poniższy kod.

```
01: import java.util.ArrayList;
02: import java.util.Collection;
03:
04: public class SafeCollection<E> {
05:
06:     private Collection<E> c = new ArrayList<E>();
07:
08:     public synchronized void add(E e) {
09:         c.add(e);
10:     }
11:
12:     public synchronized void remove(E e) {
13:         c.remove(e);
14:     }
15:
16:     @Override
17:     public String toString() {
18:         return c.toString();
19:     }
20: }
```

Które z poniższych stwierdzeń są prawdziwe?

a)	Klasa SafeCollection jest bezpieczna dla wątków ponieważ obie metody, które modyfikują jej stan (add() i remove()) są synchronizowane
b)	Klasa SafeCollection nie jest bezpieczna dla wątków. Gdyby „c” zostało zadeklarowane jako „final” wówczas SafeCollection byłaby bezpieczna dla wątków.
c)	Klasa SafeCollection nie jest bezpieczna dla wątków. Gdyby „c” zostało zadeklarowane jako: `private Collection<E> c = new Vector<E>();` wówczas SafeCollection byłaby bezpieczna dla wątków.

d)	Klasa SafeCollection nie jest bezpieczna dla wątków. Gdyby metoda toString() została zadeklarowana jako:
	`@Override`
	`public synchronized String toString() {`
	`  return c.toString();`
	`}`
	wówczas SafeCollection byłaby bezpieczna dla wątków.

■

Zadanie#3. Czy wątek zwalnia wszystkie blokady po wywołaniu na nim metody `sleep()`?

a)	Tak
b)	Nie

■

Zadanie#4. Przeanalizuj poniższy kod.

```
01:
02: class MyThread extends Thread {
03:     public static void main(String [] args) {
04:         MyThread t = new MyThread();
05:         t.start();
06:         System.out.print("one. ");
07:         t.start();
08:         System.out.print("two. ");
09:     }
10:     public void run() {
11:         System.out.print("Thread ");
12:     }
13: }
```

Wskaż możliwy wynik/wyniki wykonania powyższego programu.

a)	Compile time error
b)	Runtime error
c)	Thread one. Thread two.
d)	Wynik jest niemożliwy do przewidzenia.

■

Zadanie#5. Przeanalizuj poniższy kod.

```
01: class AcquisitionThread extends Thread {
02:     private Semaphore s;
03:
04:     public AcquisitionThread(Semaphore s) {
05:         this.s = s;
06:     }
07:     public void run() {
08:         s.tryAcquire();
09:         System.out.println(s.availablePermits());
10:     }
11: }
12:
13: public class Test {
14:
15:     public static void main(String[] args) {
16:
17:         final Semaphore s = new Semaphore(2); // 2 available permits
18:
19:         new AcquisitionThread(s).start();
20:         new AcquisitionThread(s).start();
21:         new AcquisitionThread(s).start();
22:
23:     }
24: }
```

Wskaż możliwy wynik/wyniki wykonania powyższego programu.

a)	Kod nie zostanie skompilowany z powodu nieprzechwycenia wyjątku `InterruptedException` w linii 8
b)	Zostanie wyrzucony wyjątek IllegalArgumentException z linii 21
c)	Kod wykona się poprawnie

d)	Kod wyświetli następujący wynik na konsoli: 1 0 0

■

Zadanie#6. Wskaż, która/które z poniższych konstrukcji jest/są poprawne.

a)	`Runnable r = new Runnable() {};`
b)	`Runnable r = new Runnable((public void run(){}));`
c)	`Runnable r = new Runnable() {public void run(){}};`
d)	`System.out.println(new Runnable {public void run(){}});`
e)	`Runnable r = new Runnable(public void run(){}) {};`
f)	Żadne z powyższych

■

Zadanie#7. Czy słowo kluczowe `synchronized` może być użyte do synchronizacji całej klasy jak poniżej?

```
1: synchronized class A {
2:    //class content
3: }
```

a)	Tak
b)	Nie

■

Zadanie#8. Różnica pomiędzy procesem a wątkiem polega na tym, że:

a)	Procesy nie współdzielą pamięci natomiast wątki współdzielą pamięć
b)	Wątki nie współdzielą pamięci natomiast procesy współdzielą pamięć
c)	Procesy wykonują się szybciej niż wątki
d)	Wątki wykonują się szybciej niż procesy
e)	Żadne z powyższych

■

Zadanie#9. Przeanalizuj poniższy kod.

```
01: private Map<String, String> records = new HashMap<String, String>();
02:
03: ...
04: private String getOrCreate(String id) {
05:
06:     synchronized (this) {
07:         String rec = records.get(id);
08:         if (rec == null) {
09:             rec = id;
10:             records.put(id, rec);
11:         }
12:         return rec;
13:     }
14: }
```

Czy zamieniając `HashMap` na `ConcurrentHashMap` i usuwając słowo kluczowe `synchronized` można wciąż zagwarantować synchronizację powyższego kodu?

a)	Tak
b)	Nie

■

Zadanie#10. Przeanalizuj poniższy kod.

```
01: class MyThread extends Thread {
02:   public static synchronized void main(String[] args)
03:       throws InterruptedException {
04:     Thread t = new Thread();
05:     t.start();
06:     System.out.print("X");
07:     t.wait(1000);
08:     System.out.print("Y");
09:   }
10: }
```

Wskaż, jaki będzie wynik wykonania powyższego kodu?

a)	Wyświetli X i wyjdzie
b)	Wyświetli X i nigdy nie wyjdzie
c)	Wyświetli XY i wyjdzie niemal natychmiast
d)	Wyświetli XY z 1-sekundowym opóźnieniem pomiędzy X a Y
e)	Wyświetli XY z 1000-sekundowym opóźnieniem pomiędzy X a Y
f)	Błąd kompilacji
g)	Wyjątek w trakcie uruchomienia

■

Zadanie#11. Przeanalizuj poniższy kod.

```
01: public class WaitTest {
02:   public static void main(String [] args) {
03:     System.out.print("1 ");
04:     synchronized(args) {
05:       System.out.print("2 ");
06:       try {
07:         args.wait();
08:       }
09:       catch(InterruptedException e){}
10:     }
11:     System.out.print("3 ");
12:   }
13: }
```

Wskaż, jaki będzie wynik wykonania powyższego kodu.

a)	Kompilacja nie powiedzie się ponieważ brak jest obsługi wyjątku `IllegalMonitorStateException` metody `wait()` w linii 7.

b)	`1 2 3`
c)	`1 3`
d)	`1 2`
e)	W trakcie działanie zostanie wyrzucony wyjątek `IllegalMonitorStateException` jako konsekwencja wywołania metody `wait()`.
f)	Kompilacja nie powiedzie się ponieważ konieczna jest synchronizacja na obiekcie klasy `WaitTest`.

■

Zadanie#12. W stosunku do zmiennej zadeklarowanej z użyciem słowa kluczowego `volatile` należy zapewnić synchronizację dla wszystkich wątków uzyskujących do niej dostęp.

a)	Tak
b)	Nie

■

Zadanie#13. Która klasa zawiera metody faktoryzujące umożliwiające tworzenie prekonfigurowanych instancji `ExecutorService`?

a)	`Executor`
b)	`Executors`
c)	`ExecutorService`
d)	`ExecutorServiceFactory`

■

Zadanie#14. Przeanalizuj poniższy kod.

```
01: private Integer executeTask( ExecutorService service,
02:                              Callable<Integer> task ) {
03:     // insert here
04: }
```

Wskaż, które z poniższych fragmentów wstawione zamiast komentarza poprawnie zastosują `ExecutorService` do wykonania `Callable` i zwrócą wynik `Callable` (proszę wskazać wszystkie poprawne)?

a)
```
try {
   return service.submit(task);
}
catch (Exception e) {
   return null;
}
```

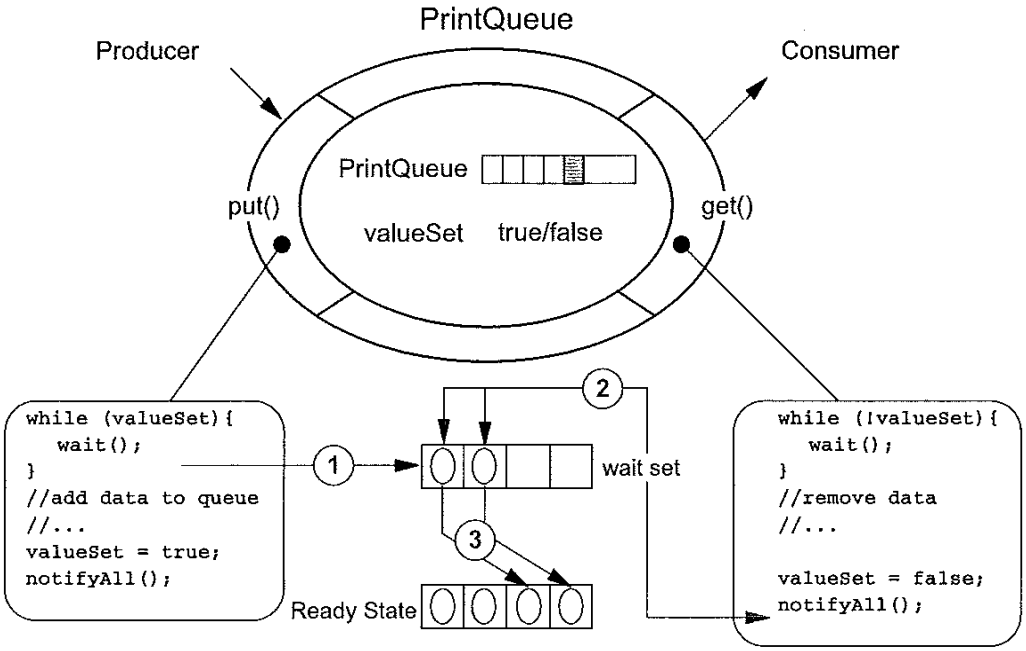
b)
```
try {
   return service.execute(task);
}
catch (Exception e) {
   return null;
}
```

c)
```
try {
   Future<Integer> future = service.submit(task);
   return future.get();
}
catch (Exception e) {
   return null;
}
```

d)
```
try {
   Result<Integer> result = service.submit(task);
   return result.get();
}
catch (Exception e) {
   return null;
}
```

■

Zadanie#15. Zaproponuj wielowątkową implementację dla poniżej zilustrowanego problemu Producenta/Konsumenta wykorzystującą wskazany mechanizm koordynacji wątków.



■

Poprawnych odpowiedzi	Ocena
<0>	n/k
<1, 8>	2.0
<9>	3.0
<10, 11>	3.5
<12, 13>	4.0
<14>	4.5
<15>	5.0