ПРАКТИЧНА РОБОТА 6
Програмування Java-додатків з підтримкою багатопоточності
Система оцінювання

№	Тема	К-ть балів
1.	*Захист принаймні одного завдання з роботи*	1
2.	Завдання на практичну роботу	3,6*
3.	*Здача звіту*	0,4
	Всього	**5**

* – діє бонусна система

1. *0,3 бала* ***(Створення та завершення потоків)*** Напишіть програму, яка створить два потоки, кожен з яких спатиме випадкову кількість секунд, а потім буде виводити повідомлення, яке включатиме ID потоку.

2. *0,6 бала* ***(Набридливі звуки).*** Класи **Timer** і **TimerTask** з пакету **java.util** дозволяють планувати запуск завдання на визначений час у майбутньому. Ви можете створити потік, що виконує у фоновому режимі і очікує заданий час. Коли час закінчиться, завдання, пов'язане з цим потоком, буде запущене. За допомогою параметрів можна запланувати завдання на повторний запуск або на запуск за певною датою. Вам не потрібно створювати потік за допомогою класу Thread, так як таймер спрощує це завдання.
   Враховуйте, що таймер виконується в своєму потоці і повинен не задіювати UI-елементи, які виконуються в своєму потоці. Для вирішення цієї проблеми можете використовувати метод runOnUiThread() для поновлення даних у компонентів.
   Класи Timer і TimerTask працюють у зв'язці. Клас Timer використовується для планування виконання задач. Запланованe до виконання завдання повинно бути екземпляром класу TimerTask. Спочатку створюється об'єкт класу TimerTask, а потім планується його запуск за допомогою класу Taimer.
   Клас TimerTask реалізує інтерфейс Runnabe і може бути використаний для створення потоку виконання.
   У класі TimerTask є абстрактний метод run(), який слід переозначити. Метод повинен містити виконуваний код.
   Метод cancel() перериває завдання і повертає значення true, якщо виконання завдання перервано.
   Метод scheduleExecutionTime() повертає час, на яке останній раз планувався запуск завдання.
   Як тільки завдання створено, воно планується для виконання об'єктом класу Timer. Як тільки об'єкт класу **Timer** створено, запуск планується за допомогою виклику його методу **schedule()**.
   Для виклику біпера використовуйте метод Toolkit.getDefaultToolkit().beep().

   Потрібно створити власний клас на базі TimerTask, що дозволятиме виконувати виклик біпера через певні проміжки часу, наприклад, 1 секунду. Даний клас може бути окремим або внутрішнім (допоміжним) класом для класу, що інкапсулює роботу з біпером (запуск таймеру, відтворення звуку та ін.).

3. *0,6 бала* Ви реалізуєте контролер для пристрою, доступ до якого відбувається за допомогою наступного інтерфейсу:

```
class Device {
    public void startup() { ... }
    public void shutdown() { ... }
}
```

Також присутні 2 датчики (нагріву та тиску), які можна використовувати для моніторингу пристрою.

```
class Sensor extends Thread {
    Device device;
    private int value;

    public Sensor(Device d) {
        device = d;
    }

    public int getValue() {
        return value;
    }

    public void updateValue() { ... }

    public void run() { ... }
}
```

Напишіть клас Controller, який може опитувати датчики в багатопоточному режимі. Реалізуйте його метод run() так, щоб він запускав пристрій, а потім стежив за новими значеннями датчиків. Контролер відключає пристрій, якщо температурний датчик повертає значення понад 70ºC, а датчик тиску – значення 100. Також завершіть метод run() у класі Sensor, який неперервно викликає updateValue() та сигналізує контролеру, якщо значення змінились.

4. *0,6 бала* ***(CountdownLatch)*** Змоделюйте роботу додатку, який запускає N потоків, що перевірять зовнішні системи та відзвітують про це засуву (latch) в головному, запускаючому класі. Як тільки всі служби верифіковані та перевірені, завантаження додатку продовжиться.

Нехай потрібно зібрати звіти для додатку, що перевірятиме показники здоров’я людини. Для цього створіть абстрактний базовий клас, який буде батьківським для усіх перевірок зовнішніх служб (мережі, бази даних, кешу тощо). Зокрема, перевірка надходження даних з мережі може описуватись класом NetworkHealthChecker:

```java
public class NetworkHealthChecker extends BaseHealthChecker
{
    public NetworkHealthChecker (CountDownLatch latch)
    {
        super("Network Service", latch);
    }

    @Override
    public void verifyService()
    {
        System.out.println("Checking " + this.getServiceName());
        try
        {
            Thread.sleep(7000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        System.out.println(this.getServiceName() + " is UP");
    }
}
```

Головний клас, який контролює запуск додатку, має ініціалізувати засув та очікувати, поки всі служби не будуть перевірені.

5. *0,6 бала* ***(CyclicBarrier)*** Припустимо, що потрібно написати програмне забезпечення для приготування страви. Наприклад, у замовленні в Mcdonalds працюватиме 3 потоки: з приготування бургеру, картоплі фрі та коли. Проте замовлення буде готове до подачі лише тоді, коли будуть готові всі його компоненти. Створіть циклічний бар'єр на 3 завдання (кожне у своєму потоці), для якого бар'єром буде Runnable-об'єкт, який сповіщатиме про готовність замовлення. Запустіть приготування замовлення та виведіть на екран інформацію щодо порядку запуску та приготування компонентів замовлення.

6. *0,6 бала* ***(Semaphore)*** Уявіть кабіну з 4 банкоматами, за допомогою яких відбувається обслуговування населення. Це означає, що в кожний момент часу може обслуговуватись максимум 4 людини. Створіть 6 потоків (людей у спільній черзі), кожен з яких працюватиме зі спільним семафором. Потік має виконувати 4 операції, кожна з яких сповіщатиме про її виконання та спатиме 1 секунду. Також потік повинен завершувати роботу (сповіщати про звільнення блокування (lock)).
Вивід програми має бути подібним до такого:

```
Total available Semaphore permits : 4
A : acquiring lock...
D : acquiring lock...
C : acquiring lock...
B : acquiring lock...
B : available Semaphore permits now: 4
C : available Semaphore permits now: 4
E : acquiring lock...
F : acquiring lock...
F : available Semaphore permits now: 2
F : got the permit!
F : is performing operation 1, available Semaphore permits : 1
D : available Semaphore permits now: 4
A : available Semaphore permits now: 4
D : got the permit!
D : is performing operation 1, available Semaphore permits : 0
```

```
E : available Semaphore permits now: 2
C : got the permit!
B : got the permit!
C : is performing operation 1, available Semaphore permits : 0
B : is performing operation 1, available Semaphore permits : 0
F : is performing operation 2, available Semaphore permits : 0
D : is performing operation 2, available Semaphore permits : 0
C : is performing operation 2, available Semaphore permits : 0
B : is performing operation 2, available Semaphore permits : 0
F : is performing operation 3, available Semaphore permits : 0
D : is performing operation 3, available Semaphore permits : 0
C : is performing operation 3, available Semaphore permits : 0
B : is performing operation 3, available Semaphore permits : 0
F : is performing operation 4, available Semaphore permits : 0
D : is performing operation 4, available Semaphore permits : 0
C : is performing operation 4, available Semaphore permits : 0
B : is performing operation 4, available Semaphore permits : 0
D : is performing operation 5, available Semaphore permits : 0
F : is performing operation 5, available Semaphore permits : 0
B : is performing operation 5, available Semaphore permits : 0
C : is performing operation 5, available Semaphore permits : 0

D : releasing lock...
F : releasing lock...
D : available Semaphore permits now: 1
A : got the permit!
A : is performing operation 1, available Semaphore permits : 0
F : available Semaphore permits now: 1
E : got the permit!


E : is performing operation 1, available Semaphore permits : 0
B : releasing lock...
B : available Semaphore permits now: 1
C : releasing lock...
C : available Semaphore permits now: 2
A : is performing operation 2, available Semaphore permits : 2
E : is performing operation 2, available Semaphore permits : 2
A : is performing operation 3, available Semaphore permits : 2
E : is performing operation 3, available Semaphore permits : 2
A : is performing operation 4, available Semaphore permits : 2
E : is performing operation 4, available Semaphore permits : 2
A : is performing operation 5, available Semaphore permits : 2
E : is performing operation 5, available Semaphore permits : 2
A : releasing lock...
A : available Semaphore permits now: 3
E : releasing lock...
E : available Semaphore permits now: 4
```

7. *0,3 бала* Перепишіть реалізацію попередньої задачі на основі мьютекса. Порівняйте результати виводу та напишіть роз’яснення щодо роботи програми.