PROGRAMOWANIE WSPÓŁBIEŻNE

SPRAWOZDANIE

Piotr Szabelski 87109

WCY23IY2S1

Zadanie nr: **PW-14**

Język implementacji: **Java: {Java, Kotlin, Scala}, .net {C#, F#}, Go, propozycja studenta**
Środowisko implementacyjne: **JetBrains**
Termin wykonania: **ostatnie zajęcia**

Zasadnicze wymagania:

a. liczba procesów sekwencyjnych powinna być dobrana z wyczuciem tak, aby zachować czytelność interfejsu i jednocześnie umożliwić zobrazowanie reprezentatywnych przykładów,
b. kod źródłowy programu musi być tak skonstruowany, aby można było „swobodnie" modyfikować liczbę procesów sekwencyjnych (z wyjątkiem zadań o ściśle określonej liczbie procesów),
c. graficzne zobrazowanie działania procesów współbieżnych,
d. odczyt domyślnych danych wejściowych ze sformatowanego, tekstowego pliku danych (yaml, json, inne),
e. [opcjonalnie] możliwość modyfikacji danych wejściowych poprzez GUI.

Sprawozdanie (w formie elektronicznej) powinno zawierać następujące elementy:

1) stronę tytułową,
2) numer i niniejszą treść zadania,
3) syntetyczny opis problemu – w tym wszystkie przyjęte założenia,
4) wykaz współdzielonych zasobów,
5) wykaz wyróżnionych punktów synchronizacji,
6) wykaz obiektów synchronizacji,
7) wykaz procesów sekwencyjnych,

Powyższe ilustrować adekwatnymi sekcjami kodu źródłowego programu.

Problem do rozwiązania:

UI.

Założenia:

Rój pszczół liczy początkowo N osobników. W danym chwili w ulu może znajdować się co najwyżej K (K<N/2) pszczół. Pszczoła, która chce dostać się do wnętrza ula, musi przejść przez jedno z dwóch istniejących wejść. Wejścia te są bardzo wąskie, więc możliwy jest w nich jedynie ruch w jedną stronę w danej chwili czasu. Zbyt długotrwałe przebywanie w ulu grozi przegrzaniem, dlatego każda z pszczół instynktownie opuszcza ul po pewnym skończonym czasie. Znajdująca się w ulu królowa, co pewien czas składa jaja, z których wylęgają się nowe pszczoły. Aby jaja zostały złożone przez królową, w ulu musi istnieć wystarczająca ilość miejsca. (1 jajo liczone jest tak jak 1 dorosły osobnik). Napisz program pszczoły "robotnicy" i królowej, tak by zasymulować cykl życia roju pszczół. Każda z pszczół "robotnic" umiera po pewnym określonym czasie X, liczonym ilością odwiedzin w ulu.

Syntetyczny opis problemu:

Problem główny:

Celem zadania jest stworzenie symulacji życia roju pszczół, gdzie rój początkowo liczy N osobników. W danej chwili w ulu może znajdować się maksymalnie K osobników, gdzie K < N/2. Pszczoły robotnice muszą przejść przez jedno z dwóch istniejących wejść do ula. Wejścia te są bardzo wąskie, dlatego możliwy jest w nich ruch tylko w jedną stronę w danej chwili. Zbyt długie przebywanie w ulu grozi przegrzaniem, więc pszczoły robotnice opuszczają ul po pewnym skończonym czasie. Królowa pszczół regularnie składa jaja, pod warunkiem że w ulu jest wystarczająco dużo miejsca. Każde jajo zajmuje jedno miejsce w ulu, tak samo jak dorosła pszczoła. Po pewnym czasie z jaj wykluwają się pszczoły robotnice. Pszczoły robotnice mają określoną liczbę odwiedzin w ulu przed śmiercią.

Przyjęte założenia:

1. Ograniczenia populacyjne:

- **Początkowa liczba pszczół (N) jest większa niż dwukrotność maksymalnej liczby pszczół w ulu (K):** To zapewnia, że w ulu nigdy nie będzie zbyt wiele pszczół, wymuszając naturalne rotacje i zapobiegając zatłoczeniu
- **Warunek K < N/2** gwarantuje, że zawsze istnieje znacząca populacja pszczół poza ulem, symulując naturalne zachowanie pszczół

2. Synchronizacja ruchu w wejściach:

- **Ruch w wejściach jest jednokierunkowy i zsynchronizowany:** Pszczoły mogą wchodzić i wychodzić z ula tylko w jedną stronę jednocześnie dla każdego z przejść, co zapobiega kolizjom w wąskich wejściach

3. Mechanizm zarządzania czasem w ulu:

- **Pszczoły opuszczają ul po określonym czasie:** Aby uniknąć przegrzania, pszczoły instynktownie wychodzą z ula po pewnym czasie (4 sekundy pracy)
- **Zbieranie nektaru:** Czas między wizytami w ulu jest losowy (1-3 sekund), symulując różne warunki zbierania pożywienia (jak cos można zakomentować jak tego nie ma być)

4. Reprodukcja i składanie jaj:

- **Królowa składa jaja tylko wtedy, gdy jest miejsce:** Zapewnia to, że liczba pszczół w ulu nie przekracza maksymalnej pojemności
- **Jajo zajmuje jedno miejsce w ulu:** Każde jajo zajmuje tyle samo miejsca co dorosła pszczoła, co wpływa na maksymalną liczbę pszczół w ulu
- **Cykliczne składanie:** Królowa składa jaja regularnie co 2 sekundy, pod warunkiem dostępności miejsca
- **Automatyczne wyklucie:** Po 1 sekundzie jajo automatycznie przekształca się w nową pszczołę robotnice

5. Cykl życia pszczół robotniczych:

- **Ograniczona liczba wizyt:** Każda pszczoła ma z góry określoną maksymalną liczbę wejść do ula przed śmiercią

- **Naturalna rotacja populacji:** Po wyczerpaniu limitu wizyt pszczoła umiera, zwalniając miejsce dla nowych pszczół

Wykaz współdzielonych zasobów:

1. Ul (pojemność ula)

Opis: Ograniczona przestrzeń w ulu, maksymalnie K pszczół może przebywać jednocześnie
kod:

```
private final Semaphore miejscaWulu;
private final int maksymalnaLiczbaPszczolWulu;
this.miejscaWulu = new Semaphore(maksymalnaLiczbaPszczolWulu);
// Funkcja wejścia - zajęcie zasobu
public void wejsciDoUla(Pszczola pszczola) {
    miejscaWulu.acquire(); // Zajęcie miejsca
}
// Funkcja wyjścia - zwolnienie zasobu
public void wyjdzZula(Pszczola pszczola) {
    miejscaWulu.release(); // Zwolnienie miejsca
}
// Składanie jaj - zajęcie zasobu
public void zlozJaja() {
    miejscaWulu.acquire(); // Jajo zajmuje miejsce
}
// Śmierć pszczoły - zwolnienie zasobu
public void zgon(Pszczola pszczola) {
    miejscaWulu.release(); // Martwa pszczoła zwalnia miejsce
}
```

2. Kolejki wejścia i wyjścia

Opis: Bufory dla pszczół oczekujących na animację ruchu przez wąskie przejścia

kod:

```
// Ul.java - Definicja kolejek
private final BlockingQueue<Pszczola> kolejkaWejscia = new LinkedBlockingQueue<>();
private final BlockingQueue<Pszczola> kolejkaWyjscia = new LinkedBlockingQueue<>();

// Dodawanie do kolejek
public void wejsciDoUla(Pszczola pszczola) {
    if (!kolejkaWejscia.offer(pszczola)) {
       System.err.println("BŁĄD: Nie udało się dodać pszczoły do kolejki wejścia");
    }
}
public void wyjdzZula(Pszczola pszczola) {
    if (!kolejkaWyjscia.offer(pszczola)) {
```

```
            System.err.println("BŁĄD: Nie udało się dodać pszczoły do kolejki wyjścia");
        }
    }
    // Worker threads - obsługa kolejek
    private void uruchomWatkiRobotnicze() {
        scheduler.execute(() -> {
            while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
                    Pszczola pszczola = kolejkaWejscia.take(); // Pobranie z kolejki
                    poruszPszczoleWejscie(pszczola, 1);
            }
        });
    }
```

3. Panel graficzny (Pane)

Opis: Obszar GUI do wyświetlania pszczół, jaj i elementów wizualnych

kod:

```
// Ul.java - Panel jako współdzielony zasób
private final Pane panel;
// Dodawanie jaj do panelu
Platform.runLater(() -> {
    if (!zatrzymanieZadane) {
        panel.getChildren().add(jajo);
    }
});
// Usuwanie zmarłych pszczół
public void zgon(Pszczola pszczola) {
    Platform.runLater(() -> panel.getChildren().remove(pszczola.getBee()));
}
// Pszczola.java - Dodawanie pszczoły do panelu
if(wykluta) {
        Platform.runLater(() -> {
        panel.getChildren().add(pszczola);
        });
}

// PszczolaKrolowa.java - Dodawanie królowej
Platform.runLater(() -> {
    panel.getChildren().add(krolowa);
});
```

4. Lista jaj w ulu

Opis: Kolekcja przechowująca aktualne jaja złożone przez królową

kod:

```
// Ul.java - Thread-safe lista jaj
private final List<Circle> jaja = new CopyOnWriteArrayList<>();
```

```
// Dodawanie jaja
public void zlozJaja() {
    Circle jajo = new Circle(5, Color.WHITE);
       jaja.add(jajo); // Dodanie do współdzielonej listy
}
// Wyklucie - usunięcie jaja
private void nowaPszczola(Circle jajo, int X, int Y) {
    Platform.runLater(() -> {
       panel.getChildren().remove(jajo);
       jaja.remove(jajo); // Usunięcie ze współdzielonej listy
    });
}
// Czyszczenie przy zatrzymaniu
public void usunJaja() {
    Platform.runLater(() -> {
       for (Circle jajo : jaja) {
              panel.getChildren().remove(jajo);
       }
       jaja.clear();
       });
}
```

5. Lista wątków pszczół

Opis: Rejestr aktywnych wątków pszczół dla zarządzania cyklem życia

kod:

```
// Ul.java - Lista wątków
private final List<Thread> watkiPszczol = new CopyOnWriteArrayList<>();
// main.java - Lista w głównej klasie
private final List<Thread> watkiPszczol = new ArrayList<>();
// Dodawanie nowego wątku pszczoły
private void nowaPszczola(Circle jajo, int X, int Y) {
    Thread watekPszczoly = new Thread(() -> new Pszczola(this, maxWejscDoUla, panel,
true, X, Y).run());
       watkiPszczol.add(watekPszczoly); // Dodanie do współdzielonej listy
    watekPszczoly.start();
}
// Zatrzymywanie wszystkich wątków
public void requestStop() {
    for (Thread watekPszczoly : watkiPszczol) {
       watekPszczoly.interrupt(); // Przerwanie wątków
    }
    watkiPszczol.clear();
}
```

6. ExecutorService (Scheduler)

Opis: Pool wątków do zarządzania zadaniami czasowymi i worker threads

kod:

```java
// Ul.java - Współdzielony scheduler
private final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(8);
// Uruchamianie worker threads
private void uruchomWatkiRobotnicze() {
        scheduler.execute(() -> { /* worker wejścia */ });
        scheduler.execute(() -> { /* worker wyjścia */ });
}
 // Planowanie zadań
public void zgon(Pszczola pszczola) {
    scheduler.schedule(() -> { /* zadanie śmierci */ }, 1, TimeUnit.SECONDS);
}
public void zlozJaja() {
        scheduler.schedule(() -> nowaPszczola(jajo, X, Y), 1, TimeUnit.SECONDS);
}
// Zamykanie
public void requestStop() {
        scheduler.shutdown();
}
```

7. Flagi kontrolne

Opis: Zmienne sterujące stanem symulacji współdzielone między wątkami

kod:

```java
// Ul.java - Flagi volatile
private volatile boolean zatrzymanieZadane = false;
private volatile boolean robotnicyDzialaja = true;
// main.java - Flaga symulacji
private boolean czySymulacjaUruchomiona = false;
// Kontrola worker threads
while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
        // logika worker thread
}
// Kontrola animacji
if (zatrzymanieZadane) {
    stop();
    return;
}
```

Wykaz wyróżnionych punktów synchronizacji

1. Wejście pszczoły do ula

Opis: Synchronizacja dostępu do ograniczonej przestrzeni ula (max K pszczół)

kod:

```
// Ul.java - Punkt synchronizacji wejścia
public void wejsciDoUla(Pszczola pszczola) {
    try {
        // PUNKT SYNCHRONIZACJI: Oczekiwanie na miejsce w ulu
        miejscaWulu.acquire();

        int zajete = maksymalnaLiczbaPszczolWulu - miejscaWulu.availablePermits();
        System.out.println("WEJSCIE: Pszczola wchodzi do ula. W ulu: " +
                zajete + "/" + maksymalnaLiczbaPszczolWulu);

        // Dodaj do kolejki wejścia
        if (!kolejkaWejscia.offer(pszczola)) {
                System.err.println("BŁĄD: Nie udało się dodać pszczoły do kolejki
wejścia");
        }
        } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        }
}
// Pszczola.java - Oczekiwanie na zakończenie animacji wejścia
wejscieLatch = new CountDownLatch(1);
ul.wejsciDoUla(this);
wejscieLatch.await(); // PUNKT SYNCHRONIZACJI: Czekaj na animację
```

2. Wyjście pszczoły z ula

Opis: Synchronizacja zwolnienia miejsca i animacji wyjścia

kod:

```
// Ul.java - Punkt synchronizacji wyjścia
public void wyjdzZula(Pszczola pszczola) {
    // Dodaj do kolejki wyjścia
    if (!kolejkaWyjscia.offer(pszczola)) {
        System.err.println("BŁĄD: Nie udało się dodać pszczoły do kolejki wyjścia");
    }
    // PUNKT SYNCHRONIZACJI: Natychmiastowe zwolnienie miejsca
    miejscaWulu.release();
    int zajete = maksymalnaLiczbaPszczolWulu - miejscaWulu.availablePermits();
    System.out.println("WYJSCIE: Pszczola wychodzi z ula. W ulu: " +
                zajete + "/" + maksymalnaLiczbaPszczolWulu);
}

// Pszczola.java - Oczekiwanie na zakończenie animacji wyjścia
wyjscieLatch = new CountDownLatch(1);
ul.wyjdzZula(this);
wyjscieLatch.await(); // PUNKT SYNCHRONIZACJI: Czekaj na animację
```

3. Synchronizacja animacji z logiką biznesową

Opis: Koordynacja między animacjami ruchu a zmianami stanu pszczół

kod:

```
// Ul.java - Zakończenie animacji i powiadomienie
public void poruszajPszczolaDoPunktow(Pszczola pszczola, double[] targetsX, double[]
targetsY, int funkcja, int trasa) {
    AnimationTimer timer = new AnimationTimer() {
        @Override
        public void handle(long now) {
               Platform.runLater(() -> {
               // ... logika animacji ...
               if (Math.abs(centerX - targetX) < 1 && Math.abs(centerY - targetY) < 1) {
                      if (indeks.incrementAndGet() >= targetsX.length) {
                     // PUNKT SYNCHRONIZACJI: Animacja zakończona
                      if (funkcja == 1) {
                             pszczola.wejscie(); // Powiadom o wejściu
                      } else {
                             pszczola.wyjscie(); // Powiadom o wyjściu
                      }
                      stop();
                      }
               }
               });
        }
    };
}
// Pszczola.java - Metody synchronizacyjne
public void wejscie() {
    if (wejscieLatch != null) {
        wejscieLatch.countDown(); // PUNKT SYNCHRONIZACJI: Odblokuj czekający wątek
    }
}
public void wyjscie() {
        if (wyjscieLatch != null) {
        wyjscieLatch.countDown(); // PUNKT SYNCHRONIZACJI: Odblokuj czekający wątek
    }
}
```

4. Synchronizacja worker threads z kolejkami

Opis: Koordynacja między dodawaniem do kolejek a ich przetwarzaniem

kod:

```
// Ul.java - Worker threads - punkty synchronizacji
private void uruchomWatkiRobotnicze() {
    // Worker dla wejść
    scheduler.execute(() -> {
        while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
               try {
               // PUNKT SYNCHRONIZACJI: Blokujące pobranie z kolejki
               Pszczola pszczola = kolejkaWejscia.take();
                poruszPszczoleWejscie(pszczola, 1);
               } catch (InterruptedException e) {
               Thread.currentThread().interrupt();
               break;
               }
        }
```

```
        });
        // Worker dla wyjść
        scheduler.execute(() -> {
        while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
                try {
                // PUNKT SYNCHRONIZACJI: Blokujące pobranie z kolejki
                Pszczola pszczola = kolejkaWyjscia.take();
                poruszPszczoleWyjscie(pszczola, 2);
                } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                break;
                }
        }
    });
}
```

5. Składanie jaj przez królową

Opis: Synchronizacja dostępu królowej do ula i zarządzanie miejscem

kod:

```
// Ul.java - Synchronizacja składania jaj
public void zlozJaja() {
    try {
        // PUNKT SYNCHRONIZACJI: Królowa czeka na miejsce w ulu
        miejscaWulu.acquire();

        int zajete = maksymalnaLiczbaPszczolWulu - miejscaWulu.availablePermits();
        System.out.println("JAJO: Krolowa sklada jajo. W ulu: " +
                zajete + "/" + maksymalnaLiczbaPszczolWulu);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        return;
    }
    // PUNKT SYNCHRONIZACJI: Sprawdzenie flagi zatrzymania
    if (zatrzymanieZadane) {
        miejscaWulu.release();
        return;
    }
    // Planowanie wyklucia
    scheduler.schedule(() -> nowaPszczola(jajo, X, Y), 1, TimeUnit.SECONDS);
}
// PszczolaKrolowa.java - Cykliczne składanie
@Override
public void run() {
    while (true) {
        try {
                Thread.sleep(2000);
                ul.zlozJaja(); // PUNKT SYNCHRONIZACJI: Wywołanie składania
        } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                break;
          }
    }
```

```
}
```

6. Synchronizacja śmierci pszczoły

Opis: Koordynacja usuwania pszczoły i zwolnienia zasobów

kod:

```
// Ul.java - Synchronizacja śmierci
public void zgon(Pszczola pszczola) {
    scheduler.schedule(() -> {
        // PUNKT SYNCHRONIZACJI: Usunięcie z GUI w wątku JavaFX
        Platform.runLater(() -> panel.getChildren().remove(pszczola.getBee()));
        // PUNKT SYNCHRONIZACJI: Zwolnienie miejsca w ulu
        miejscaWulu.release();
        int dostepneMiejsca = miejscaWulu.availablePermits();
        int zajete = maksymalnaLiczbaPszczolWulu - dostepneMiejsca;
        System.out.println("SMIERC: Pszczola zginela. W ulu: " +
                zajete + "/" + maksymalnaLiczbaPszczolWulu);
    }, 1, TimeUnit.SECONDS);
}
// Pszczola.java - Wywołanie śmierci
if(i == maxWejscDoUla-1) {
    ustawMartwaPszczole();
    ul.zgon(this); // PUNKT SYNCHRONIZACJI: Zgłoszenie śmierci
    break;
}
```

7. Synchronizacja zatrzymania symulacji

Opis: Koordynacja zamykania wszystkich wątków i czyszczenia zasobów

kod:

```
// Ul.java - Synchronizacja zatrzymania
public void requestStop() {
    // Ustawienie flag zatrzymania
    zatrzymanieZadane = true;
    robotnicyDzialaja = false;
    // Przerwanie wątków pszczół
    for (Thread watekPszczoly : watkiPszczol) {
        watekPszczoly.interrupt();
        }
    watkiPszczol.clear();
    // PUNKT SYNCHRONIZACJI: Zamknięcie scheduler
    scheduler.shutdown();
        try {
        if (!scheduler.awaitTermination(2, TimeUnit.SECONDS)) {
                scheduler.shutdownNow();
        }
        } catch (InterruptedException e) {
        scheduler.shutdownNow();
        Thread.currentThread().interrupt();
```

```
        }
}
// main.java - Synchronizacja zatrzymania głównego
private void zatrzymajSymulacje() {
    czySymulacjaUruchomiona = false;

    // PUNKT SYNCHRONIZACJI: Przerwanie wątku królowej
    if (watekKrolowej != null) {
        watekKrolowej.interrupt();
    }
    // PUNKT SYNCHRONIZACJI: Przerwanie wątków pszczół
    for (Thread watekPszczoly : watkiPszczol) {
        watekPszczoly.interrupt();
        }
    watkiPszczol.clear();
    // PUNKT SYNCHRONIZACJI: Zatrzymanie ula
    if (ul != null) {
        ul.requestStop();
        ul.usunJaja();
    }
}
```

8. Synchronizacja z wątkiem JavaFX

Opis: Koordynacja aktualizacji GUI z wątkami logiki biznesowej

kod:

```
// Pszczola.java - Synchronizacja GUI
Platform.runLater(() -> {
    // PUNKT SYNCHRONIZACJI: Bezpieczna aktualizacja GUI
    pszczola.setX(X - 12.5);
    pszczola.setY(Y - 12.5);
        panel.getChildren().add(pszczola);
});
Platform.runLater(() -> {
        if (pszczola != null && obrazekNormalnejPszczoly != null) {
        pszczola.setImage(obrazekNormalnejPszczoly);
    }
});
// Ul.java - Synchronizacja GUI dla jaj
Platform.runLater(() -> {
    if (!zatrzymanieZadane) {
        panel.getChildren().add(jajo);
        jajo.setCenterX(X);
        jajo.setCenterY(Y);
    }
});
// PszczolaKrolowa.java - Synchronizacja GUI królowej
Platform.runLater(() -> {
    krolowa.setX(150 - 55);
    krolowa.setY(350 - 55);
    panel.getChildren().add(krolowa);
});
```

Wykaz obiektów synchronizacji

1. Semaphore - kontrola pojemności ula

Opis: Ogranicza dostęp do ula maksymalnie K pszczół jednocześnie

kod:

```
// Ul.java - Definicja semafora
private final Semaphore miejscaWulu;
// Konstruktor - inicjalizacja semafora
public Ul(int maksymalnaLiczbaPszczolWulu, Pane panel, int maxWejscDoUla) {
	this.miejscaWulu = new Semaphore(maksymalnaLiczbaPszczolWulu);
}
// Zajęcie miejsca - wejście do ula
public void wejsciDoUla(Pszczola pszczola) {
	try {
	miejscaWulu.acquire(); // Zajęcie semafora
	} catch (InterruptedException e) {
	Thread.currentThread().interrupt();
	}
}
// Zwolnienie miejsca - wyjście z ula
public void wyjdzZula(Pszczola pszczola) {
	miejscaWulu.release(); // Zwolnienie semafora
}
// Zajęcie miejsca - składanie jaja
public void zlozJaja() {
	try {
	miejscaWulu.acquire(); // Jajo zajmuje miejsce
	} catch (InterruptedException e) {
	Thread.currentThread().interrupt();
	}
}
// Zwolnienie miejsca - śmierć pszczoły
public void zgon(Pszczola pszczola) {
	miejscaWulu.release(); // Martwa pszczoła zwalnia miejsce
}
```

2. CountDownLatch - synchronizacja animacji wejścia

Opis: Synchronizuje wątek pszczoły z zakończeniem animacji wejścia do ula

kod:

```
// Pszczola.java - Definicja CountDownLatch
private CountDownLatch wejscieLatch;
// Utworzenie nowego latch przed wejściem
wejscieLatch = new CountDownLatch(1);
ul.wejsciDoUla(this);
wejscieLatch.await(); // Czekaj na odblokowanie
```

```
// Metoda wywoływana po zakończeniu animacji wejścia
public void wejscie() {
	if (wejscieLatch != null) {
	wejscieLatch.countDown(); // Odblokuj czekający wątek
	}
}
```

3. CountDownLatch - synchronizacja animacji wyjścia

Opis: Synchronizuje wątek pszczoły z zakończeniem animacji wyjścia z ula

kod:

```
// Pszczola.java - Definicja CountDownLatch
private CountDownLatch wyjscieLatch;
// Utworzenie nowego latch przed wyjściem
wyjscieLatch = new CountDownLatch(1);
ul.wyjdzZula(this);
wyjscieLatch.await(); // Czekaj na odblokowanie
// Metoda wywoływana po zakończeniu animacji wyjścia
public void wyjscie() {
	if (wyjscieLatch != null) {
	wyjscieLatch.countDown(); // Odblokuj czekający wątek
	}
}
```

4. BlockingQueue - kolejka wejścia do ula

Opis: Thread-safe kolejka pszczół oczekujących na animację wejścia

kod:

```
// Ul.java - Definicja BlockingQueue
private final BlockingQueue<Pszczola> kolejkaWejscia = new LinkedBlockingQueue<>();
// Dodawanie do kolejki (producer)
public void wejsciDoUla(Pszczola pszczola) {
	if (!kolejkaWejscia.offer(pszczola)) {
	System.err.println("BŁĄD: Nie udało się dodać pszczoły do kolejki wejścia");
	}
}
// Pobieranie z kolejki (consumer) - worker thread
private void uruchomWatkiRobotnicze() {
	scheduler.execute(() -> {
	while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
		try {
		Pszczola pszczola = kolejkaWejscia.take(); // Blokujące pobranie
		poruszPszczoleWejscie(pszczola, 1);
		} catch (InterruptedException e) {
		Thread.currentThread().interrupt();
		break;
		}
	}
	});
```

```
}
```

5. BlockingQueue - kolejka wyjścia z ula

Opis: Thread-safe kolejka pszczół oczekujących na animację wyjścia

kodu:

```
// Ul.java - Definicja BlockingQueue
private final BlockingQueue<Pszczola> kolejkaWyjscia = new LinkedBlockingQueue<>();
// Dodawanie do kolejki (producer)
public void wyjdzZula(Pszczola pszczola) {
	if (!kolejkaWyjscia.offer(pszczola)) {
	System.err.println("BŁĄD: Nie udało się dodać pszczoły do kolejki wyjścia");
	}
}
// Pobieranie z kolejki (consumer) - worker thread
private void uruchomWatkiRobotnicze() {
	scheduler.execute(() -> {
	while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
		try {
		Pszczola pszczola = kolejkaWyjscia.take(); // Blokujące pobranie
		poruszPszczoleWyjscie(pszczola, 2);
		} catch (InterruptedException e) {
		Thread.currentThread().interrupt();
		break;
		}
	}
	});
}
```

6. ScheduledExecutorService - zarządzanie wątkami

Opis: Pool wątków do wykonywania zadań czasowych i worker threads

kod:

```
// Ul.java - Definicja ScheduledExecutorService
private final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(8);
// Wykonywanie worker threads
private void uruchomWatkiRobotnicze() {
	scheduler.execute(() -> { /* worker wejścia */ });
	scheduler.execute(() -> { /* worker wyjścia */ });
}
// Planowanie zadań czasowych - wyklucie
public void zlozJaja() {
	scheduler.schedule(() -> nowaPszczola(jajo, X, Y), 1, TimeUnit.SECONDS);
}
// Planowanie zadań czasowych - śmierć
public void zgon(Pszczola pszczola) {
	scheduler.schedule(() -> {
	Platform.runLater(() -> panel.getChildren().remove(pszczola.getBee()));
	miejscaWulu.release();
```

```
        }, 1, TimeUnit.SECONDS);
}
// Synchronizowane zamknięcie
public void requestStop() {
        scheduler.shutdown();
        try {
        if (!scheduler.awaitTermination(2, TimeUnit.SECONDS)) {
                scheduler.shutdownNow();
        }
        } catch (InterruptedException e) {
        scheduler.shutdownNow();
        Thread.currentThread().interrupt();
        }
}
```

7. CopyOnWriteArrayList - thread-safe lista jaj

Opis: Thread-safe lista przechowująca aktywne jaja w ulu

kod:

```
// Ul.java - Definicja CopyOnWriteArrayList
private final List<Circle> jaja = new CopyOnWriteArrayList<>();
// Thread-safe dodawanie jaja
public void zlozJaja() {
        Circle jajo = new Circle(5, Color.WHITE);
        jaja.add(jajo); // Thread-safe dodanie
}
// Thread-safe usuwanie jaja
private void nowaPszczola(Circle jajo, int X, int Y) {
        Platform.runLater(() -> {
        panel.getChildren().remove(jajo);
        jaja.remove(jajo); // Thread-safe usunięcie
        });
}
// Thread-safe czyszczenie wszystkich jaj
public void usunJaja() {
        Platform.runLater(() -> {
        for (Circle jajo : jaja) { // Thread-safe iteracja
                panel.getChildren().remove(jajo);
        }
        jaja.clear(); // Thread-safe czyszczenie
        });
}
```

8. CopyOnWriteArrayList - thread-safe lista wątków pszczół

Opis: Thread-safe lista przechowująca aktywne wątki pszczół

kod:

```
// Ul.java - Definicja CopyOnWriteArrayList
private final List<Thread> watkiPszczol = new CopyOnWriteArrayList<>();
```

```
// Thread-safe dodawanie wątku
private void nowaPszczola(Circle jajo, int X, int Y) {
	Thread watekPszczoly = new Thread(() -> new Pszczola(this, maxWejscDoUla, panel,
true, X, Y).run());
	watkiPszczol.add(watekPszczoly); // Thread-safe dodanie
	watekPszczoly.start();
}
// Thread-safe przerwanie wszystkich wątków
public void requestStop() {
	for (Thread watekPszczoly : watkiPszczol) { // Thread-safe iteracja
	watekPszczoly.interrupt();
	}
	watkiPszczol.clear(); // Thread-safe czyszczenie
}
```

9. ArrayList - lista wątków pszczół (main)

Opis: Lista wątków pszczół w głównej klasie (nie thread-safe, ale używana tylko z głównego wątku)

kod:

```
// main.java - Definicja ArrayList
private final List<Thread> watkiPszczol = new ArrayList<>();
// Dodawanie wątków przy starcie symulacji
private void uruchomSymulacje(Pane panel) {
	for (int i = 0; i < liczbaPoczatkowychPszczol; i++) {
	Thread watekPszczoly = new Thread(() -> new Pszczola(ul, maxWejscDoUla, panel,
false, 0, 0).run());
	 watkiPszczol.add(watekPszczoly); // Dodanie do listy
	watekPszczoly.start();
	}
}
// Przerwanie wątków przy zatrzymaniu
private void zatrzymajSymulacje() {
	for (Thread watekPszczoly : watkiPszczol) {
	watekPszczoly.interrupt(); // Przerwanie wątku
	}
	watkiPszczol.clear(); // Czyszczenie listy
}
```

10. volatile boolean - flagi kontrolne

Opis: Flagi synchronizacyjne widoczne między wątkami

kodu:

```
// Ul.java - Flagi volatile
private volatile boolean zatrzymanieZadane = false;
private volatile boolean robotnicyDzialaja = true;
// Używanie flag w worker threads
private void uruchomWatkiRobotnicze() {
	scheduler.execute(() -> {
```

```
        while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
                // logika worker thread
        }
        });
}
// Kontrola animacji
public void poruszajPszczolaDoPunktow(...) {
        AnimationTimer timer = new AnimationTimer() {
        @Override
        public void handle(long now) {
                if (zatrzymanieZadane) {
                stop();
                return;
                }
        }
        };
}
// Ustawienie flag przy zatrzymaniu
public void requestStop() {
        zatrzymanieZadane = true;
        robotnicyDzialaja = false;
}
// main.java - Flaga symulacji
private boolean czySymulacjaUruchomiona = false;

private void uruchomSymulacje(Pane panel) {
        if (!czySymulacjaUruchomiona) {
         czySymulacjaUruchomiona = true;
        // logika uruchomienia
        }
}
```

11. Platform.runLater - synchronizacja z JavaFX

Opis: Mechanizm synchronizacji z wątkiem JavaFX dla aktualizacji GUI

kod:

```
// Pszczola.java - Synchronizacja GUI pszczoły
Platform.runLater(() -> {
        pszczola.setX(X - 12.5);
        pszczola.setY(Y - 12.5);
        panel.getChildren().add(pszczola);
});
Platform.runLater(() -> {
        if (pszczola != null && obrazekNormalnejPszczoly != null) {
        pszczola.setImage(obrazekNormalnejPszczoly);
        }
});
// PszczolaKrolowa.java - Synchronizacja GUI królowej
Platform.runLater(() -> {
        krolowa.setX(150 - 55);
        krolowa.setY(350 - 55);
        panel.getChildren().add(krolowa);
});
```

```
// Ul.java - Synchronizacja GUI jaj i usuwania
Platform.runLater(() -> {
        if (!zatrzymanieZadane) {
        panel.getChildren().add(jajo);
        jajo.setCenterX(X);
        jajo.setCenterY(Y);
        }
});

Platform.runLater(() -> panel.getChildren().remove(pszczola.getBee()));
```

12. AtomicInteger - atomowa pozycja jaj

Opis: Thread-safe licznik pozycji dla rozmieszczania jaj w ulu

kod:

```
// Ul.java - Definicja AtomicInteger
private final AtomicInteger zmiana = new AtomicInteger(0);
// Atomowa aktualizacja pozycji jaja
public void zlozJaja() {
        int currentPos = zmiana.getAndUpdate(pozycja -> (pozycja + 15) >= 90 ? 0 :
pozycja + 15);
    int X = 80;
        int Y = 310 + currentPos;

        // Użycie pozycji do umieszczenia jaja
        Circle jajo = new Circle(5, Color.WHITE);
        Platform.runLater(() -> {
        jajo.setCenterX(X);
        jajo.setCenterY(Y);
        });
}
```

Wykaz procesów sekwencyjnych

1. Wątek główny aplikacji (JavaFX Application Thread)

Opis: Główny wątek aplikacji odpowiedzialny za GUI i obsługę zdarzeń

kod:

```
// main.java - Główny proces sekwencyjny aplikacji
public class main extends Application {
    public static void main(String[] args) {
        launch(args); // Uruchomienie aplikacji JavaFX - główny wątek
    }
    @Override
    public void start(Stage scenaPodstawowa) {
        // Sekwencyjne tworzenie interfejsu użytkownika
        Properties wlasciwosci = new Properties();
        // ... wczytanie konfiguracji ...
```

```
        Pane panel = new Pane();
        panel.setPrefSize(1250, 700);
        // Sekwencyjne dodawanie elementów GUI
        Button przyciskStart = new Button("Start");
        Button przyciskStop = new Button("Stop");
        // ... pozostałe elementy GUI ...
        panel.getChildren().addAll(przyciskStart, przyciskStop, /* ... */);
        scenaPodstawowa.setScene(new Scene(panel, 1250, 700));
        scenaPodstawowa.show();
    }
    // Sekwencyjne metody obsługi zdarzeń (wykonywane w JavaFX Thread)
    private void uruchomSymulacje(Pane panel) {
        // Sekwencyjna walidacja danych
        if (liczbaPoczatkowychPszczol <= 0 || maksymalnaLiczbaPszczolWulu <= 0) {
            // Wyświetlenie alertu
            return;
        }
        // Sekwencyjne uruchomienie symulacji
        czySymulacjaUruchomiona = true;
        ul = new Ul(maksymalnaLiczbaPszczolWulu, panel, maxWejscDoUla);
        // Uruchomienie wątku królowej
        watekKrolowej = new Thread(() -> new PszczolaKrolowa(ul, panel).run());
        watekKrolowej.start();
    }
}
```

2. Proces pszczoły robotniczej

Opis: Sekwencyjny cykl życia pojedynczej pszczoły robotniczej

kod:

```
// Pszczola.java - Proces sekwencyjny pszczoły robotniczej
@Override
public void run() {
    Random random = new Random();
    // Sekwencyjny cykl życia pszczoły
    for (int i = 0; i < maxWejscDoUla; i++) {
        try {
            // 1. Specjalny przypadek - pierwsza iteracja wykluteej pszczoły
            if(wykluta && i == 0) {
                // Sekwencyjne wykonanie animacji pierwnego lotu
                Random rand = new Random();
                double x = 20 + Math.random() * 430 - 12.5;
                double y = rand.nextBoolean() ? 20 + Math.random() * 280 - 12.5
                                              : 400 + Math.random() * 270 - 12.5;

                // Animacja lotu do pozycji
                Platform.runLater(() -> {
                    Timeline timeline = new Timeline();
                    KeyFrame keyFrame = new KeyFrame(Duration.seconds(1),
                        new KeyValue(pszczola.xProperty(), x),
                        new KeyValue(pszczola.yProperty(), y));
                    timeline.getKeyFrames().add(keyFrame);
```

```
                    timeline.play();
                });
                wejscie(); // Oznacz jako w ulu
                Thread.sleep(4000); // Praca w ulu
                ustawCzerwonaPszczole(); // Zmiana wyglądu
                // Wyjście z ula
                wyjscieLatch = new CountDownLatch(1);
                ul.wyjdzZula(this);
                wyjscieLatch.await();
                ustawNormalnaPszczole();
            }
            // 2. Sekwencyjny cykl normalny
            // Faza zbierania nektaru (poza ulem)
            Thread.sleep(1000 + random.nextInt(4000)); // 1-5 sekund zbierania
            // Faza wejścia do ula
            wejscieLatch = new CountDownLatch(1);
            ul.wejsciDoUla(this); // Próba wejścia
            wejscieLatch.await(); // Czekaj na zakończenie animacji wejścia
            ustawNormalnaPszczole(); // Pszczoła w ulu
            // 3. Sprawdzenie końca życia
            if(i == maxWejscDoUla-1) {
                ustawMartwaPszczole(); // Śmierć
                ul.zgon(this);
                break; // Koniec procesu
            }
            // 4. Faza pracy w ulu i wyjścia
            Thread.sleep(4000); // Praca w ulu
            ustawCzerwonaPszczole(); // Zmiana wyglądu na wyjście
            // Wyjście z ula
            wyjscieLatch = new CountDownLatch(1);
            ul.wyjdzZula(this);
            wyjscieLatch.await(); // Czekaj na zakończenie animacji wyjścia
            ustawNormalnaPszczole(); // Powrót do normalnego wyglądu

        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            Platform.runLater(() -> panel.getChildren().remove(pszczola));
            break; // Przerwanie procesu
        }
    }
}
```

3. Proces pszczoły królowej

Opis: Sekwencyjny proces składania jaj przez królową

kod:

```
// PszczolaKrolowa.java - Proces sekwencyjny królowej
@Override
public void run() {
    // Nieskończony sekwencyjny cykl składania jaj
    while (true) {
        try {
```

```
            // 1. Oczekiwanie na kolejny cykl
            Thread.sleep(2000); // Interwał 2 sekundy między jajami
            // 2. Próba złożenia jaja
            ul.zlozJaja(); // Sekwencyjne wywołanie składania jaja
        } catch (InterruptedException e) {
            // 3. Obsługa przerwania - zakończenie procesu
            Thread.currentThread().interrupt();
            // Sekwencyjne usunięcie królowej z panelu
            Platform.runLater(() -> panel.getChildren().remove(krolowa));
            break; // Koniec procesu królowej
        }
    }
}
// Konstruktor - sekwencyjne inicjowanie królowej
public PszczolaKrolowa(Ul ul, Pane panel) {
    this.ul = ul;
    this.panel = panel;
    // Sekwencyjne wczytanie obrazka
    try {
        this.obrazekKrolowej = new Image("file:src/resources/krolowa.png");
    } catch (Exception e) {
        System.err.println("BŁĄD: Nie można wczytać obrazka królowej!");
        throw new RuntimeException("Brak pliku obrazka królowej", e);
    }
    // Sekwencyjne utworzenie i dodanie do GUI
    this.krolowa = new ImageView(obrazekKrolowej);
    krolowa.setFitWidth(100);
    krolowa.setFitHeight(100);
    Platform.runLater(() -> {
        krolowa.setX(150 - 55);
        krolowa.setY(350 - 55);
        panel.getChildren().add(krolowa);
    });
}
```

4. Worker thread - proces obsługi wejść

Opis: Sekwencyjny proces obsługi kolejki wejść do ula

kod:

```
// Ul.java - Worker thread dla wejść
private void uruchomWatkiRobotnicze() {
    // Proces sekwencyjny worker thread wejścia
    scheduler.execute(() -> {
        System.out.println("WORKER WEJSCIA: Uruchomiony");
        // Nieskończona pętla sekwencyjna
        while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
            try {
                // 1. Sekwencyjne oczekiwanie na pszczołę
                Pszczola pszczola = kolejkaWejscia.take(); // Blokujące pobranie

                // 2. Sekwencyjne przetworzenie - animacja wejścia
                poruszPszczoleWejscie(pszczola, 1); // Trasa 1 dla wejścia
```

```
            } catch (InterruptedException e) {
                // 3. Obsługa przerwania
                Thread.currentThread().interrupt();
                break; // Zakończenie procesu worker
            }
        }
        System.out.println("WORKER WEJSCIA: Zatrzymany");
    });
}
```

5. Worker thread - proces obsługi wyjść

Opis: Sekwencyjny proces obsługi kolejki wyjść z ula

kod:

```
// Ul.java - Worker thread dla wyjść
private void uruchomWatkiRobotnicze() {
    // Proces sekwencyjny worker thread wyjścia
    scheduler.execute(() -> {
        System.out.println("WORKER WYJSCIA: Uruchomiony");
        // Nieskończona pętla sekwencyjna
        while (robotnicyDzialaja && !Thread.currentThread().isInterrupted()) {
            try {
                // 1. Sekwencyjne oczekiwanie na pszczołę
                Pszczola pszczola = kolejkaWyjscia.take(); // Blokujące pobranie
                // 2. Sekwencyjne przetworzenie - animacja wyjścia
                poruszPszczoleWyjscie(pszczola, 2); // Trasa 2 dla wyjścia
            } catch (InterruptedException e) {
                // 3. Obsługa przerwania
                Thread.currentThread().interrupt();
                break; // Zakończenie procesu worker
            }
        }
        System.out.println("WORKER WYJSCIA: Zatrzymany");
    });
}
```

6. Proces animacji ruchu pszczoły

Opis: Sekwencyjny proces animacji przemieszczania pszczoły przez punkty kontrolne

kod:

```
// Ul.java - Proces sekwencyjny animacji
public void poruszajPszczolaDoPunktow(Pszczola pszczola, double[] targetsX, double[]
targetsY, int funkcja, int trasa) {
    if (zatrzymanieZadane || pszczola == null) return;

    AtomicInteger indeks = new AtomicInteger(0);
    ImageView obrazekPszczoly = pszczola.getBee();
```

```
    // Sekwencyjny proces animacji
    AnimationTimer timer = new AnimationTimer() {
        private double targetX = targetsX[0]; // Pierwszy punkt docelowy
        private double targetY = targetsY[0];
        // Obliczenie prędkości
        private double currentCenterX = obrazekPszczoly.getX() +
obrazekPszczoly.getFitWidth()/2;
        private double currentCenterY = obrazekPszczoly.getY() +
obrazekPszczoly.getFitHeight()/2;
        private double velocityX = (targetX - currentCenterX) / 70;
        private double velocityY = (targetY - currentCenterY) / 70;
        @Override
        public void handle(long now) {
            if (zatrzymanieZadane) {
                stop();
                return;
            }
            Platform.runLater(() -> {
                // 1. Sekwencyjne przemieszczenie
                obrazekPszczoly.setX(obrazekPszczoly.getX() + velocityX);
                obrazekPszczoly.setY(obrazekPszczoly.getY() + velocityY);
                // 2. Sprawdzenie dotarcia do celu
                double centerX = obrazekPszczoly.getX() +
obrazekPszczoly.getFitWidth()/2;
                double centerY = obrazekPszczoly.getY() +
obrazekPszczoly.getFitHeight()/2;

                if (Math.abs(centerX - targetX) < 1 && Math.abs(centerY - targetY) < 1)
{
                    // 3. Sekwencyjne przejście do następnego punktu
                    if (indeks.incrementAndGet() < targetsX.length) {
                        // Kolejny punkt w sekwencji
                        targetX = targetsX[indeks.get()];
                        targetY = targetsY[indeks.get()];

                        // Przeliczenie prędkości
                        currentCenterX = obrazekPszczoly.getX() +
obrazekPszczoly.getFitWidth()/2;
                        currentCenterY = obrazekPszczoly.getY() +
obrazekPszczoly.getFitHeight()/2;
                        velocityX = (targetX - currentCenterX) / 70;
                        velocityY = (targetY - currentCenterY) / 70;
                    } else {
                        // 4. Sekwencyjne zakończenie animacji
                        if (funkcja == 1) {
                            pszczola.wejscie(); // Powiadom o wejściu
                        } else {
                            pszczola.wyjscie(); // Powiadom o wyjściu
                        }
                        stop(); // Koniec procesu animacji
                    }
                }
            });
        }
    };
    timer.start();}
```

7. Proces wyklucia pszczoły

Opis: Sekwencyjny proces przekształcenia jaja w nową pszczołę

kod:

```
// Ul.java - Proces sekwencyjny wyklucia
private void nowaPszczola(Circle jajo, int X, int Y) {
    if (zatrzymanieZadane) return;
    // Sekwencyjny proces wyklucia
    Platform.runLater(() -> {
        // 1. Usunięcie jaja z GUI
        panel.getChildren().remove(jajo);
        jaja.remove(jajo);
        // 2. Monitoring stanu ula
        int aktualnyStanPrint = maksymalnaLiczbaPszczolWulu -
miejscaWulu.availablePermits();
        // 3. Sekwencyjne utworzenie nowej pszczoły
        Thread watekPszczoly = new Thread(() -> new Pszczola(this, maxWejscDoUla, panel,
true, X, Y).run());
        watkiPszczol.add(watekPszczoly); // Dodanie do rejestru
        watekPszczoly.start(); // Uruchomienie procesu pszczoły
    });
}
// Proces składania jaja (wywołuje wyklucie)
public void zlozJaja() {
    try {
        // 1. Sekwencyjne zajęcie miejsca
        miejscaWulu.acquire();
        // 2. Monitoring i logging
        int zajete = maksymalnaLiczbaPszczolWulu - miejscaWulu.availablePermits();
        System.out.println("JAJO: Krolowa sklada jajo. W ulu: " + zajete + "/" +
maksymalnaLiczbaPszczolWulu);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
        return;
    }
    if (zatrzymanieZadane) {
        miejscaWulu.release();
        return;
    }
    // 3. Sekwencyjne utworzenie jaja
    int currentPos = zmiana.getAndUpdate(pozycja -> (pozycja + 15) >= 90 ? 0 : pozycja +
15);
    int X = 80;
    int Y = 310 + currentPos;
    Circle jajo = new Circle(5, Color.WHITE);
    jaja.add(jajo);
    // 4. Dodanie do GUI
    Platform.runLater(() -> {
        if (!zatrzymanieZadane) {
            panel.getChildren().add(jajo);
            jajo.setCenterX(X);
            jajo.setCenterY(Y);
            jajo.setStroke(Color.BLACK);
```

```
            jajo.setStrokeWidth(1);
        }
    });
    // 5. Zaplanowanie wyklucia (po 1 sekundzie)
    scheduler.schedule(() -> nowaPszczola(jajo, X, Y), 1, TimeUnit.SECONDS);
}
```

8. Proces zatrzymania symulacji

Opis: Sekwencyjny proces zamykania wszystkich wątków i czyszczenia zasobów

kod:

```
// Ul.java - Sekwencyjny proces zatrzymania
public void requestStop() {
    // 1. Sekwencyjne ustawienie flag
    zatrzymanieZadane = true;
    robotnicyDzialaja = false;
    // 2. Sekwencyjne przerwanie wątków pszczół
    for (Thread watekPszczoly : watkiPszczol) {
        watekPszczoly.interrupt();
    }
    watkiPszczol.clear();
    // 3. Sekwencyjne zamknięcie schedulera
    scheduler.shutdown();
    try {
        if (!scheduler.awaitTermination(2, TimeUnit.SECONDS)) {
            scheduler.shutdownNow(); // Wymuszenie zatrzymania
        }
    } catch (InterruptedException e) {
        scheduler.shutdownNow();
        Thread.currentThread().interrupt();
    }
}
// main.java - Sekwencyjny proces zatrzymania głównego
private void zatrzymajSymulacje() {
    // 1. Sekwencyjne ustawienie flagi
    czySymulacjaUruchomiona = false;
    // 2. Sekwencyjne przerwanie wątku królowej
    if (watekKrolowej != null) {
        watekKrolowej.interrupt();
    }
    // 3. Sekwencyjne przerwanie wątków pszczół
    for (Thread watekPszczoly : watkiPszczol) {
        watekPszczoly.interrupt();
    }
    watkiPszczol.clear();
    // 4. Sekwencyjne zatrzymanie ula
    if (ul != null) {
        ul.requestStop();
        ul.usunJaja();
    }
}
```