**SPRAWOZDANIE**

**PROGRAMOWANIE WSPÓŁBIEŻNE**

**PROJEKT**

**Joanna Łuniewska WCY23IY2S1**

**Treść zadania:**

Zadanie nr: PW-15

W pewnym miasteczku znajduje się kompleks basenów krytych. W jego skład wchodzą: basen olimpijski, basen rekreacyjny i brodzik dla dzieci. Na każdym z nich może przebywać w danej chwili maksymalnie X1, X2 i odpowiednio X3 osób płacących. Osoba odwiedzająca centrum kupuje bilet czasowy, który upoważnia ją do korzystania z dowolnego basenu zgodnie z regulaminem pływalni przez pewien określony czas. Dzieci poniżej 10. Roku życia nie płacą za bilet. Istnieje pewna liczba osób VIP posiadająca karnety wstępu, które umożliwiają wejście na pływalnie z pominięciem kolejki oczekujących. Chętni do wejścia na basen, w różnym wieku ( od 1 roku do 70 lat ), pojawiają się w losowy sposób co pewien czas. Okresowo centrum jest zamykane, aby umożliwić obsłudze wymianę wody we wszystkich basenach równocześnie. W trakcie tej czynności nikt nie ma prawa przebywać na terenie pływalni ( należy poczekać aż wszyscy opuszczą basen). Napisz programy kasjera i klientów, który zagwarantuje sprawną obsługę i korzystanie z obiektu zgodnie z niżej podanym regulaminem. Podpowiedź: Zaimplementuj dziecko i jego opiekuna jako jeden proces.   
Regulamin pływalni:

-tylko osoby pełnoletnie mogą korzystać z basenu olimpijskiego

-dzieci poniżej 10. roku życia nie mogą przebywać na basenie bez opieki osoby dorosłej

-w brodziku kąpać mogą się jedynie dzieci do 5. roku życia i ich opiekunowie

-dzieci do 3. Roku życia muszą pływać w Pampersach J.

-średnia wieku w basenie rekreacyjnym nie może być większa niż 40 lat.

-noszenie czepków nie jest obowiązkowe

**Opis problemu:**

Celem projektu jest stworzenie **symulacji centrum basenowego** z wieloma klientami (wątkami), którzy odwiedzają różne typy basenów zgodnie z ich cechami osobowymi (wiek, VIP, opiekun, pampers). Centrum działa w cyklach otwarcia i zamknięcia, w których następuje również zmiana wody, a klienci odpowiednio reagują na dostępność obiektu.

**Założenia:**

Typy basenów:

* **Olimpijski** – dla dorosłych (np. wiek > 18), jednocześnie może przebywać 5 osób.
* **Rekreacyjny** – dla młodzieży i dorosłych, średnia wieku klientów musi być mniejsza od 40. Pojemność to 4 osoby.
* **Brodzik** – tylko dla dzieci poniżej 5 lat, z pampersami i z opiekunem. Maksymalna liczba osób to 3.
* Liczbę klientów i maksymalną liczbę osób na basenie można zmieniać w pliku config.json

Klient (wątek):

* Kupuje bilet u kasjera.
* Czeka na otwarcie centrum.
* Przechodzi do odpowiedniego basenu – jeśli nie może wejść, czeka aż zwolni się miejsce.
* Spędza na basenie **10 sekund**, a następnie opuszcza go.

Kasjer:

* Obsługuje kolejkę klientów tylko, gdy centrum jest otwarte.
* Osoby przychodzące, gdy centrum jest zamknięte, czekają w kolejce oczekującej.

Centrum**:**

* Otwiera się i zamyka co **20 sekund**.
* Po zamknięciu czeka, aż wszyscy klienci opuszczą baseny, a następnie przeprowadza „zmianę wody” (symulacja).
* GUI wizualizuje aktualny stan centrum, kolejki oraz zawartość każdego basenu.

GUI (Swing):

* Aktualizuje na bieżąco:
  + Status centrum (OTWARTE, ZAMKNIĘTE – klienci wychodzą, ZAMKNIĘTE – zmiana wody),
  + Kolejkę do kasjera,
  + Stan każdego basenu (liczba osób + szczegóły klientów).

**Wykaz współdzielonych zasobów i ich ochrona:**

**Zasób**: osobyNaBasenie **typ:** ArrayList<Osoba>   
Występuje w klasie **Basen** i jej podklasach. Chroniony przez **synchronized** w metodach **wejdź**, **opuscBasen** i **czyPusty**.

Wiele osób-klientów może próbować jednocześnie wejść i wyjść z basenu.

Kod, w którym jest to zawarte (klasa **Basen**):

*public synchronized boolean wejdz(Osoba osoba) {  
 if (czyMoznaWejsc(osoba) && osobyNaBasenie.size() < pojemnosc) {  
 osobyNaBasenie.add(osoba);  
 aktualizujGUI();  
 return true;  
 }  
 return false;  
}  
  
public synchronized void opuscBasen(Osoba osoba) {  
 osobyNaBasenie.remove(osoba);  
 aktualizujGUI();  
}*

**Zasób**: kolejkaNormalna, kolejkaOczekujaca **typ:** List<Osoba>  
Występuje w klasie **Kasjer**. Ochrona przez **synchronized** na obiekcie Kasjer.

Listy są modyfikowane zarówno przez metodę kupBilet, która jest wywoływana przez klientów i przez wątek Kasjer.

Części kodu, w którym jest to zawarte (klasa **Kasjer**):

*public void kupBilet(Osoba osoba) {  
 synchronized (this) {  
 if (centrum != null && !centrum.isOtwarte()) {  
  
 if (osoba.isVIP()) {  
 // szukanie VIP-a w kolejce  
 int index = 0;  
 for (int i = 0; i < kolejkaOczekujaca.size(); i++) {  
 if (kolejkaOczekujaca.get(i).isVIP()) {  
 index = i + 1;  
 } else {  
 break;  
 }  
 }  
 kolejkaOczekujaca.add(index, osoba);  
 } else {  
 kolejkaOczekujaca.add(osoba);  
 }  
 gui.aktualizujKolejke(new ArrayList<>(kolejkaOczekujaca));  
  
 System.out.println("Osoba " + osoba.nrOosby + " czeka do otwarcia centrum.");  
 } else {  
  
 if (osoba.isVIP()) {  
 int index = 0;  
 for (int i = 0; i < kolejkaNormalna.size(); i++) {  
 if (kolejkaNormalna.get(i).isVIP()) {  
 index = i + 1;  
 } else {  
 break;  
 }  
 }  
 kolejkaNormalna.add(index, osoba);  
 } else {  
 kolejkaNormalna.add(osoba);  
 }  
 gui.aktualizujKolejke(new ArrayList<>(kolejkaNormalna));  
 notify(); // obudź kasjera  
 }  
 }  
}*

**Zasób**: otwarte **typ:** boolean  
Występuje w klasie **Centrum**. Chroniony jest przez **synchronized** w metodach **otworz**, **zamknij**, **isOtwarte**.

Jest to flaga stanu centrum. Zmieniana przez zamknij i otworz, odczytywana przez klientów i kasjera. Synchronizacja zapewnia spójność stanu.

Części kodu, w którym jest to zawarte (klasa **Kasjer**):

*public synchronized void otworz() {*

*otwarte = true;*

*}*

*public synchronized void zamknij() {*

*otwarte = false;*

*}*

*public synchronized boolean isOtwarte() {*

*return otwarte;*

*}*

**Zasób**: maBilet(pole w klasie Osoba) **typ:** boolean  
Wykorzystywany jest w wątkach **Klient, Kasjer**. Chroniony jest przez **synchronized** na obiekcie **osoba.**

Kasjer ustawia to pole, a klient czeka na jego zmianę. Synchronizacja odbywa się poprzez **wait()** i **notify()** na obiekcie **osoba**.

Części kodu, w którym jest to zawarte (klasy **Kasjer i Klient**):

*try{  
 kasjer.kupBilet(osoba);  
 synchronized(osoba)  
 {  
 while(!osoba.maBilet){  
 osoba.wait();  
 }  
 }  
  
}catch(InterruptedException e){  
 System.out.println("Przerwano");  
}*

*synchronized (osoba) {  
 osoba.maBilet = true;  
 osoba.notify();  
 //wypisanie w konsoli  
 System.out.println("Kasjer sprzedał bilet osobie " + osoba.nrOosby + " Dane: wiek " + osoba.wiek+", VIP: "+ osoba.isVIP()+ ", ma Opiekuna: " + osoba.maOpiekuna);  
}*

**Zasób**: komponenty GUI (kolejkaPanel, statusCentrum, paneleBasenow) **typ:** Swing GUI  
Występuje w klasie **CentrumGUI**. Ochrona odbywa się przez **SwingUtilities.invokeLater().**

Wątki inne niż GUI aktualizują komponenty, więc muszą używać SwingUtilities.invokeLater() by zapewnić bezpieczeństwo wątku EDT.

Części kodu, w którym jest to zawarte (klasa **CentrumGUI**):

*public void aktualizujStatus(int otwarte) {  
 SwingUtilities.invokeLater(() -> {  
 String status;  
 switch (otwarte) {  
 case 1:  
 status = "OTWARTE";  
 break;  
 case 2:  
 status = "ZAMKNIĘTE – klienci wychodzą";  
 break;  
 case 3:  
 status = "ZAMKNIĘTE – zmiana wody";  
 break;  
 default:  
 status = "NIEZNANY";  
 }  
 statusCentrum.setText("Status centrum: " + status);  
 });  
  
}*

**Punkty synchronizacji (słowo kluczowe synchronized):**

* 1. Metoda **basen.wejdz(Osoba osoba)**

Synchronizuje dostęp do listy osób na basenie, aby zapewnić spójność przy wejściu.

Przykład w kodzie:

*public synchronized boolean wejdz(Osoba osoba) {  
 if (czyMoznaWejsc(osoba) && osobyNaBasenie.size() < pojemnosc) {  
 osobyNaBasenie.add(osoba);  
 aktualizujGUI();  
 return true;  
 }  
 return false;  
}*

* 1. Metoda **basen.opuscBasen(Osoba osoba)**  
     Chroni operację usunięcia osoby z listy i aktualizacji GUI.

Przykład w kodzie:

*public synchronized void opuscBasen(Osoba osoba) {  
 osobyNaBasenie.remove(osoba);  
 aktualizujGUI();  
}*

* 1. Metoda **basen.czyPusty()**  
     Zabezpiecza dostęp do listy osób na basenie przed równoczesnym odczytem/zapisem.

Przykład w kodzie:

*public synchronized boolean czyPusty() {  
 return osobyNaBasenie.isEmpty();  
}*

* 1. Blok **w kasjer.kupBilet(Osoba osoba)**Synchronizuje dostęp do kolejek **kolejkaNormalna** i **kolejkaOczekujaca.**

Przykład w kodzie:

*public void kupBilet(Osoba osoba) {  
 synchronized (this) {  
 if (centrum != null && !centrum.isOtwarte()) {  
  
 if (osoba.isVIP()) {  
 // szukanie VIP-a w kolejce  
 int index = 0;  
 for (int i = 0; i < kolejkaOczekujaca.size(); i++) {  
 if (kolejkaOczekujaca.get(i).isVIP()) {  
 index = i + 1;  
 } else {  
 break;  
 }  
 }  
 kolejkaOczekujaca.add(index, osoba);  
 } else {  
 kolejkaOczekujaca.add(osoba);  
 }  
 gui.aktualizujKolejke(new ArrayList<>(kolejkaOczekujaca));  
  
 System.out.println("Osoba " + osoba.nrOosby + " czeka do otwarcia centrum.");  
 } else {  
  
 if (osoba.isVIP()) {  
 int index = 0;  
 for (int i = 0; i < kolejkaNormalna.size(); i++) {  
 if (kolejkaNormalna.get(i).isVIP()) {  
 index = i + 1;  
 } else {  
 break;  
 }  
 }  
 kolejkaNormalna.add(index, osoba);  
 } else {  
 kolejkaNormalna.add(osoba);  
 }  
  
 gui.aktualizujKolejke(new ArrayList<>(kolejkaNormalna));  
  
  
 notify(); // obudź kasjera  
 }  
 }  
}*

* 1. Metoda **kasjer.wznowObsluge()**Synchronizuje przenoszenie osób z oczekującej do głównej kolejki.

Przykład w kodzie:

*public void wznowObsluge() {  
 synchronized (this) {  
 kolejkaNormalna.addAll(kolejkaOczekujaca);  
 kolejkaOczekujaca.clear();  
 notify(); // obudź kasjera jeśli czeka  
 }  
}*

* 1. Blok w **kasjer.run()**Synchronizuje pobieranie osoby z kolejki i czekanie na zgłoszenie nowej.

Przykład w kodzie:

*synchronized (this) {  
 while (kolejkaNormalna.isEmpty()) {  
 try {  
 wait();  
 } catch (InterruptedException e) {  
 return;  
 }  
 }  
 osoba = kolejkaNormalna.get(0);  
}*

* 1. Blok synchronized(osoba) w **klient.run()**Osoba czeka na bilet, aż zostanie powiadomiona przez kasjera.

Przykład w kodzie:

*try{  
 kasjer.kupBilet(osoba);  
 synchronized(osoba)  
 {  
 while(!osoba.maBilet){  
 osoba.wait();  
 }  
 }  
  
}catch(InterruptedException e){  
 System.out.println("Przerwano");  
}*

* 1. Blok synchronized (centrum) w **klient.run()**Klient czeka aż centrum zostanie otwarte.

Przykład w kodzie:

*synchronized (centrum) {  
 while (!centrum.isOtwarte()) {  
 try {  
 centrum.wait();  
 } catch (InterruptedException e) {  
 System.out.println("Przerwano");  
 }  
 }  
}*

* 1. Metody **centrum.otworz()** i **centrum.zamknij()**Synchronizują zmianę stanu otwarcia oraz powiadamianie wątków.

Przykład w kodzie:

*public synchronized void otworz() {  
 otwarte = true;  
 System.out.println("Centrum otwarte!");  
 gui.aktualizujStatus(1);  
 notifyAll();  
  
 if (kasjer != null) {  
 kasjer.wznowObsluge();  
 }  
}  
  
public synchronized void zamknij() {  
 otwarte = false;  
 System.out.println("Centrum: Zamknięcie. Czekamy aż baseny się opróżnią...");  
 gui.aktualizujStatus(2);  
}*

**Obiekty synchronizacji (na których wykonano synchronized):**

Obiekt synchronizowany: **this** w klasie **Kasjer**  
Synchronizuje metody kupBilet(), run(), wznowObsluge().

Przykład kodu:

*public void wznowObsluge() {  
 synchronized (this) {  
 kolejkaNormalna.addAll(kolejkaOczekujaca);  
 kolejkaOczekujaca.clear();  
 notify(); // obudź kasjera jeśli czeka  
 }  
}*

Obiekt synchronizowany: **osoba (Osoba obiekt)**  
Synchronizacja oczekiwania na bilet.

Przykład kodu:

*synchronized(osoba)  
{  
 while(!osoba.maBilet){  
 osoba.wait();  
 }  
}*

Obiekt synchronizowany: **centrum (Centrum)**  
Synchronizacja oczekiwania na otwarcie.

Przykład kodu:

*synchronized (centrum) {  
 while (!centrum.isOtwarte()) {  
 try {  
 centrum.wait();  
 } catch (InterruptedException e) {  
 System.out.println("Przerwano");  
 }  
 }  
}*

Obiekt synchronizowany: **Basen (this)**  
Synchronizacja operacji wejścia/wyjścia z basenu i odczytu listy osób.

Przykład kodu ( w klasie Basen):

*public synchronized boolean wejdz(Osoba osoba) {  
 if (czyMoznaWejsc(osoba) && osobyNaBasenie.size() < pojemnosc) {  
 osobyNaBasenie.add(osoba);  
 aktualizujGUI();  
 return true;  
 }  
 return false;  
}  
  
public synchronized void opuscBasen(Osoba osoba) {  
 osobyNaBasenie.remove(osoba);  
 aktualizujGUI();  
}*

**Wykaz procesów sekwencyjnych:**

**Start aplikacji (Main)**

* Wczytanie konfiguracji → Utworzenie GUI → Start kasjera i centrum.

Przykład kodu dla wczytania konfiguracji i utworzenie GUI:

*Konfiguracja config = Konfiguracja.wczytaj("src/main/resources/config.json");  
CentrumGUI gui = new CentrumGUI();  
Kasjer kasjer = new Kasjer(gui);  
Centrum cent = new Centrum(baseny, kasjer, gui);*

Przykład kodu dla startu kasjera i centrum:

*kasjer.start();  
cent.start();*

**Tworzenie i startowanie klientów**

* Generowanie osoby → Startowanie wątku Klient.

Przykład kodu:

*Osoba osoba = GeneratorOsob.generujOsobe(osoby++);  
Klient klient = new Klient(osoba, kasjer, baseny, cent, gui);  
klient.start();*

**Wątek Klient:**

* Próba zakupu biletu → czekanie na jego otrzymanie → czekanie na otwarcie centrum → losowanie basenu → wejście, pobyt, wyjście.

*synchronized (osoba) {  
 while (!osoba.maBilet()) {  
 osoba.wait();  
 }  
}*

*while (!centrum.isOtwarte()) {  
 try {  
 centrum.wait();  
 } catch (InterruptedException e) {  
 System.out.println("Przerwano");  
 }  
}*

*ArrayList<Basen> losoweBaseny = new ArrayList<>(baseny);  
Collections.shuffle(losoweBaseny); // losowa kolejności  
  
for (Basen b : losoweBaseny) {  
 if (b.wejdz(osoba)) {  
 System.out.println(osoba.nrOosby + " weszła na " + b.nazwa);  
 znalazlBasen = true;  
 try {  
 Thread.sleep(10000); // czas spędzony na basenie  
 } catch (InterruptedException e) {  
 System.out.println("Przerwano");  
 }  
 b.opuscBasen(osoba);  
 }  
}*

**Wątek Kasjer:**

* Obsługuje kolejkę → przypisuje bilet i budzi klienta.

*synchronized (this) {*

*if (centrum != null && !centrum.isOtwarte()) {*

*if (osoba.isVIP()) {*

*// szukanie VIP-a w kolejce  
 int index = 0;  
 for (int i = 0; i < kolejkaOczekujaca.size(); i++) {*

*if (kolejkaOczekujaca.get(i).isVIP()) {*

*index = i + 1;*

*} else {*

*break;  
 }  
 }  
 kolejkaOczekujaca.add(index, osoba);  
 } else {*

*kolejkaOczekujaca.add(osoba);  
 }  
 gui.aktualizujKolejke(new ArrayList<>(kolejkaOczekujaca));  
 System.out.println("Osoba " + osoba.nrOosby + " czeka do otwarcia centrum.");*

*} else {*

*if (osoba.isVIP()) {*

*int index = 0;  
 for (int i = 0; i < kolejkaNormalna.size(); i++) {  
 if (kolejkaNormalna.get(i).isVIP()) {  
 index = i + 1;  
 } else {*

*break;  
 }  
 }  
 kolejkaNormalna.add(index, osoba);  
 } else {  
 kolejkaNormalna.add(osoba);  
 }*

*gui.aktualizujKolejke(new ArrayList<>(kolejkaNormalna));  
 notify();  
 }  
}*

**Wątek Centrum:**

* Cykl: otwarte 20s → zamknięcie → czekanie na opróżnienie → zmiana wody 5s → otwarcie → powiadomienie kasjera i klientów.

Przykład kodu:

*Thread.sleep(20000); // centrum otwarte przez 20s  
zamknij(); // zamknięcie centrum  
boolean pusto;  
do {  
 pusto = true;  
 for (Basen b : baseny) {  
 if (!b.czyPusty()) {  
 pusto = false;  
 break;  
 }  
 }  
 if (!pusto) {  
 Thread.sleep(500); // czekanie na opróżnienie  
 }  
} while (!pusto);  
System.out.println("Centrum: Wszystkie baseny puste. Zmiana wody...");*

*gui.aktualizujStatus(3);*

*Thread.sleep(5000); // zmiana wody  
otworz(); // ponowne otwarcie*