Laboratorium 5

Celem ćwiczenia jest budowa prostych aplikacji wielowątkowych działających w środowisku graficznym. Program ćwiczenia:

- uruchamianie i zatrzymywanie wątków,
- usypianie wątków na zadany czas metodą `sleep(…)`
- synchronizacja wątków przy pomocy metod synchronizowanych,
- użycie metod `wait()` i `notify()` lub `notifyAll()`.

Programy przykładowe

Program `ProducentKonsumentDemo.java` ilustruje sposób synchronizacji współbieżnych wątków symulujących jeden z popularnych klasycznych problemów współbieżności – **problem producenta-konsumenta**. W programie do synchronizacji wykorzystano metody synchronizowane oraz instrukcje `wait(),notify().`

Zadanie 1 (obowiązkowe)

Proszę przeanalizować program `ProducentKonsumentDemo.java`, a następnie rozbudować ten program tak by posiadał graficzny interfejs użytkownika, który umożliwiać będzie zmianę pojemności bufora oraz zmianę liczby producentów i konsumentów. Przebieg symulacji powinien być wyświetlany w oknie graficznym typu `JTextArea`. Ponadto program powinien być wyposażony w dodatkowe przyciski umożliwiające wstrzymywanie i uruchamianie symulacji w dowolnym momencie.

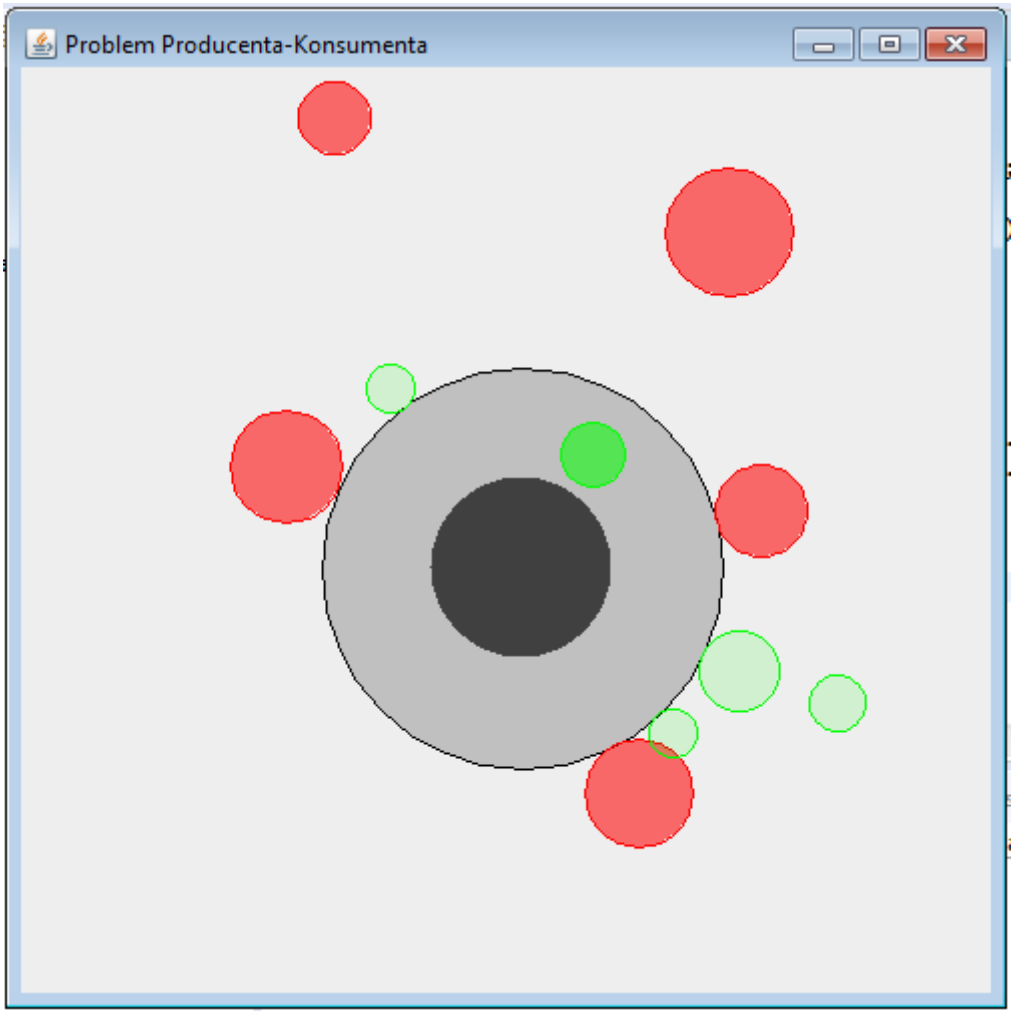
Osoby ambitne mogą rozbudować ten program tak by przebieg symulacji był prezentowany w formie prostej animacji graficznej – np. producenci i konsumenci pokazany w postaci kółek, których kolor pokazuje ich stan, natomiast bieżąca zawartość bufora wypisana w prostokącie. (Zob. program przykładowy `ProdKons2.jar`)

Zadanie 2 (dla ambitnych)

Proszę napisać program który w formie graficznej animacji będzie wizualizował problem producenta – konsumenta. Producenci i konsumenci powinni być przedstawieni w formie kółek przesuwających się płynnie po panelu. Na środku panelu w formie koła powinien być wyrysowany magazyn, w którym będą składowane produkty. Kółko reprezentujące producenta pobiera produkty przy brzegu panelu, a następnie dostarcza te produkty do magazynu. Oddanie produktów następuje wtedy, gdy kółko producenta znajdzie się całym obwodem wewnątrz magazynu. Jeśli w magazynie nie ma wystarczającej ilości miejsca, to producent przechodzi w stan oczekiwania (kółko zmienia kolor i się zatrzymuje). Podobnie kółko reprezentujące konsumenta pobiera produkty z magazynu wtedy, gdy znajdzie się wewnątrz magazynu całym obwodem. Jeśli brak wystarczającej ilości produktów, to konsument przechodzi w stan oczekiwania (kółko zmienia kolor i się zatrzymuje). Konsument oddaje produkty przy brzegu panelu.

Wewnątrz magazynu może poruszać się tylko jedno kółko reprezentujące producenta lub konsumenta. Jeśli w magazynie porusza się już jakieś kółko, to kółko chcące wjechać do magazynu musi czekać na dostęp przy obwodzie magazynu. Uwaga: Kółka reprezentujące producentów oczekujących na wolne miejsce w magazynie oraz kółka reprezentujące

konsumentów oczekujących na produkty są zatrzymane i nie blokują dostępu dla kolejnych kółek. Przykładową wizualizację problemu w tej formie przedstawia program `ProducerConsumerAmimaton.jar`



Wskazówki pomocnicze:

W programie warto zdefiniować klasę abstrakcyjną `Circle`, która będzie klasą bazową dla klas `Producer` oraz `Consumer`. W tej klasie powinny być pamiętane parametry kółka (współrzędne środka, promień, kierunek i prędkość ruchu) oraz pomocnicze metody sprawdzające, czy kółko jest przy brzegu panelu (np. `isTouchingBorder()`), czy jest całym obwodem poza magazynem (np. `isOutsideBuffer()`) oraz czy jest całym obwodem wewnątrz magazynu (np. `isInsideBuffer()`). Klasy `Producer` oraz `Consumer` powinny dziedziczyć po klasie `Circle` oraz powinny implementować interfejs `Runnable`. Do konstruktorów tych klas należy przekazywać referencję na obiekt reprezentujący wspólny magazyn, którą należy zapamiętać (np. `buffer`) Wówczas metoda `run` dla producenta może mieć schematycznie następującą postać:

```
void run(){
   while(true){

      // sekcja lokalna - kółko jest poza magazynem
      while(isOutsideBuffer()){
         if (isTouchingBorder()){ // kólko na brzegu panelu
            // ładuj produkty
            // ...
         }
         // przesuń kółko i odrysuj panel
       }
      // koniec sekcji lokalnej

      // sekcja krytyczna - ruch kółka w magazynie
       synchronized(buffer){
          do{
             if(isInsideBuffer()){ // gdy kółko jest całym obwodem wewnątrz

                while(buffer.full()){ // brak miejsca w magazynie
                   buffer.wait();
                }

                //Rozładuj produkty
                // ....
             }
             // przesuń kółko i odrysuj panel
             // ...

             }while(!isOutsideBuffer())

          buffer.notifyAll();

       } // koniec sekcji krytycznej
   }
}
```

Zasobem współdzielonym przez wszystkich producentów i konsumentów jest magazyn, którego referencja jest pamiętana w polu `buffer`. Dlatego blok `synchronized` oraz wywołania metod `wait()` i `notifyAll()` dotyczą tego pola.

W analogiczny sposób należy zdefiniować metodę `run()` dla konsumenta.