Java.util.concurrency

Example1

Pokazuje w jaki sposób utworzyć kilka wątków. Można zaobserwować że kolejność wykonywania jest niedeterministyczna.

Example2

Przykład demonstruje użycie bariery.

Example3

Pokazuje w jaki sposób kontrolować kolejność uruchamiania się wątków, zademonstrowanie oczekiwania na śmierć wątków.

Example4

Przykład pokazujący jak zabić wątki w czasie ich trwania z wykorzystaniem przerwania.

Zadanie:

1. Wykonaj sprawozdanie, opisz wymienione poniżej funkcje/klasy/interfejsy paczki java.util.concurrency odpowiadające za zrównoleglenie, użyte w przykładach.

   - interfejs Runnable

   Interfejs Runnable powinna implementować każda klasa, której obiekty mają być traktowane jako osobne wątki. Wewnątrz klasy implementującej konieczne jest nadpisanie bezargumentowej metody run(). Interfejs ten zawiera powszechne protokoły dla obiektów, które chcą wykonywać kod podczas gdy są aktywne. Aktywny oznacza wątek, który został rozpoczęty, a nie został zatrzymany. Runnable jest implementowany przez klasę Thread. Implementacja interfejsu runnable umozliwia klasie działanie w obszarze wątku bez konieczności poszerzania klasy Thread, poprzez bezpośrednie stworzenie jej instancji wewnątrz klasy. Interfejsu warto używać, gdy nie chce się nadpisywać innej metody klasy Thread oprócz run().

   - interfejs Callable<T>

   Interfejs podobny do Runnable. Służy do zwrócenia rezultatu określonego typu i ewentualnego wyrzucenia wyjątku. Przeznaczone dla klas, których instancje są obsługiwane przez wyjątki. Posiada jedną bezargumentową metodę call().

   - klasa Executor (w tym metoda newFixedThreadPool())

   Klasa ta służy do wywoływania zdefiniowanych zadań obsługiwanych przez interfejs Runnable. Zwalnia ona z konieczności obsługi pozostałych aspektów wątku takich jak sposób wykonywania , zadaniowanie w czasie. Pozwala storzyć zadanie dla wątku w nastepujący sposób:

```
for (int i=1; i<=4; i++) {
  exec.execute(new Task(String.valueOf(i)));
}
```

   Metoda newFixedThreadPool() służy do zdefiniowania liczby wątków obsługujących kolejkę zadań do wykonania. Jeżeli jakiś wątek zostanie wyłączony przez na przykład błąd pojawi się nowy jako wypełnienie. Liczba zdefiniowanych aktywnych watków pozostaje stała do momentu ich wyłączenia

- klasa ExecutorService (szczególnie metody shutdown(), shutdownNow() , awaitTermination(), isTerminated())

Zapewnia metody do przerywania i śledzenia postępu w wykonywaniu zadań asynchronicznych. Obiekt tej metody może być wyłączony co powstrzymuje go przed przyjmowanie nowych zadań do wykonania i przerwania wykonywanych. Metoda shutdown() służy do przerwania wykonywania zadań przy czym program czeka na na zakończenie zadań już zakolejkowanych i już wykonywanych. Metoda shutdownNow() usiłuje przerwać wykonywanie zadań aktualnie wykonywanych. Zapobiega także rozpoczęciu wykonywania aktualnie czekających zadań. Metoda awaitTermination() blokuje działanie przez maksymalny czas sprecyzowany jako argument do momentu, aż wszystkie zadania od momentu wywołania żądania przerwania się skończą lub przekroczony zostanie zdefiniowany czas lub też przerwany zostanie wątek w zależności od tego, które z wymienionych okoliczności wystąpią wcześniej. Metoda isTerminated() zwraca true, jeśli wszystkie zadania od momentu żądania przerwania zostały zakończone. Warunkiem zwrócenia przez metodę true jest uprzednie wywołanie metod shutdown() lub shutdownNow().

- klasa FutureTask<T> i jej metoda T

Klasa słuzy do inicjacji asynchronicznego prztwarzania. Implementuje interfejsy Callable<T> oraz Runnable. Klasa zapewnia implementację Future. Dostarcza metody do rozpoczęcia i zakończenia asynchronicznego przetwarzania, zapytań do weryfikacji aktualnego stanu procesu przetwarzania oraz pobrania rezultatów komutacji metodą get(). Pobranie rezultatu możliwe jest tylko w momencie zakończenia prztwarzania. Zakończone przetwarzanie nie może być wycofane lub zresetowane chyba, że zostanie wywołane metodą runAndReset() . W ogólności FutureTask<V> definiuj przyszłe zadanie, które wykona operacje zdefiniowane w callable<T>. Parametr<V> to typ zwracanego wyniku przetwarzania. FutureTask<V> udostępnia dodatkowo poniższe metody:

Methods

Modifier and Type	Method and Description
boolean	cancel(boolean mayInterruptIfRunning) Attempts to cancel execution of this task.
protected void	done() Protected method invoked when this task transitions to state isDone (whether normally or via cancellation).
V	get() Waits if necessary for the computation to complete, and then retrieves its result.
V	get(long timeout, TimeUnit unit) Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.
boolean	isCancelled() Returns true if this task was cancelled before it completed normally.
boolean	isDone() Returns true if this task completed.
void	run() Sets this Future to the result of its computation unless it has been cancelled.
protected boolean	runAndReset() Executes the computation without setting its result, and then resets this future to initial state, failing to do so if the computation encounters an exception or is cancelled.
protected void	set(V v) Sets the result of this future to the given value unless this future has already been set or has been cancelled.
protected void	setException(Throwable t) Causes this future to report an ExecutionException with the given throwable as its cause, unless this future has already been set or has been cancelled.

- metody Thread.sleep(), Thread.yield(), <<Thread Object>>.join()

Thread.sleep() umożliwia uśpienie aktualnie wykonywanego wątku na czas podany jako paramter w milisekundach.

Thread.yield() to rodzaj wskazówki dla procesora,zarządcy przydziału zasobów, która mówi, że aktualnie wykonywany wątek chce pozbyć się przydzielonego mu procesora.

<<Thread Object>>.join()

Metoda ta powoduje wstrzymanie programu do czasu, aż podany obiekt wątku (<<ThreadObject>>) „umrze”.

- **funkcje System.currentTimeMillis()**

  Funkcja zwraca aktualny czas w milisekundach. Czas od 1 stycznia 1970 do teraz.

- **czym różni się Catch(Exception e) od Catch(InterruptException e)**

  InterrupException e zostaje wyrzucony kiedy wątek czeka, jest uśpiony lub w inny sposób zajęty i zostaje przerwany zarówno przed jak i czasie wykonania. Metoda może tez próbować sprawdzić stan wątku i na tej rzucić wyjątek.

  Exception to zbiorczy typ dla obsługi wyjątków zarówno typu checked jak i unchecked.

Wyniki oraz program prześlij do swojego repozytorium. Umieść je w folderze o tej samie nazwie co ten PDF.