Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wykład 4.
Kolejki – implementacja, przetwarzanie

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Kolejki

Kolejka (*ang. queue*) – struktura danych o charakterze listy; ciąg elementów przechowywanych w sposób umożliwiający ich przetwarzanie w określonej kolejności.

Kolejki

Kolejka (*ang. queue*) – struktura danych o charakterze listy; ciąg elementów przechowywanych w sposób umożliwiający ich przetwarzanie w określonej kolejności.

Różnice w porównaniu do list dotyczą dostępności elementów:

- w listach możemy uzyskać dostęp do dowolnego elementu (sekwencyjnie lub indeksowo),
- w kolejce dostępny jest w danej chwili tylko jeden element, tzw. czoło (głowa), (*ang. head*); ta rola elementu wynika z implementacji kolejki.

Kolejka

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Podstawowe kategorie kolejek:
- nieograniczone (nie jest określona maksymalna liczba elementów),
- ograniczone (gdy jest określona maksymalna liczba elementów, czyli limit; konieczna jest wówczas dodatkowa metoda sprawdzająca czy kolejka jest pełna).

Podstawowe kategorie kolejek:
- nieograniczone (nie jest określona maksymalna liczba elementów),
- ograniczone (gdy jest określona maksymalna liczba elementów, czyli limit; konieczna jest wówczas dodatkowa metoda sprawdzająca czy kolejka jest pełna).

Podział ze względu na kolejność pobierania elementów z kolejki:

1. Kolejki zwykle (o kolejności pobierania z kolejki decyduje wyłącznie kolejność wstawiania); stosuje się następujące strategie pobierania:
   - FIFO (*ang. First-In, First-Out*, "pierwsze przyszło, pierwsze wyszło"), "typowa" kolejka; kolejność pobierania jest tożsama z kolejnością umieszczania elementów w kolejce).
   - LIFO (*ang. Last-In, First-Out*, "ostatnie przyszło, pierwsze wyszło").

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Podstawowe kategorie kolejek:
- nieograniczone (nie jest określona maksymalna liczba elementów),
- ograniczone (gdy jest określona maksymalna liczba elementów, czyli limit; konieczna jest wówczas dodatkowa metoda sprawdzająca czy kolejka jest pełna).

Podział ze względu na kolejność pobierania elementów z kolejki:
1. Kolejki zwykle (o kolejności pobierania z kolejki decyduje wyłącznie kolejność wstawiania); stosuje się następujące strategie pobierania:
   - FIFO (*ang. First-In, First-Out*, "pierwsze przyszło, pierwsze wyszło"), "typowa" kolejka; kolejność pobierania jest tożsama z kolejnością umieszczania elementów w kolejce).
   - LIFO (*ang. Last-In, First-Out*, "ostatnie przyszło, pierwsze wyszło").
2. Kolejki priorytetowe (o kolejności pobierania decyduje priorytet elementów kolejki; ale o tym będzie w późniejszym terminie).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

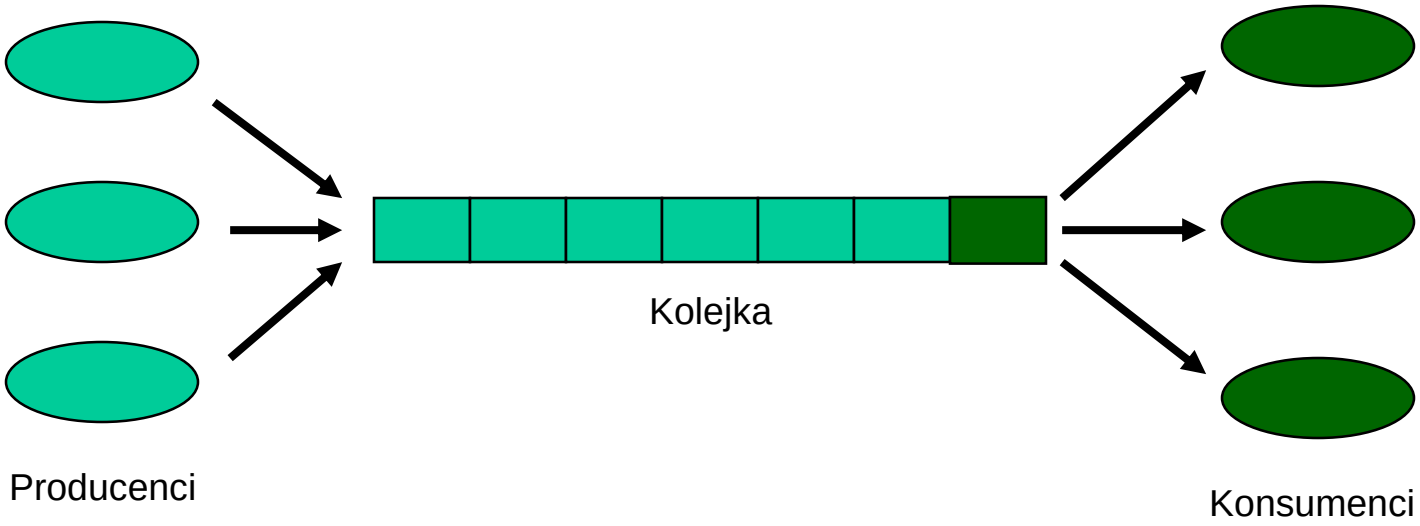
Podstawowe kategorie kolejek:
- nieograniczone (nie jest określona maksymalna liczba elementów),
- ograniczone (gdy jest określona maksymalna liczba elementów, czyli limit; konieczna jest wówczas dodatkowa metoda sprawdzająca czy kolejka jest pełna).

Podział ze względu na kolejność pobierania elementów z kolejki:
1. Kolejki zwykle (o kolejności pobierania z kolejki decyduje wyłącznie kolejność wstawiania); stosuje się następujące strategie pobierania:
   - FIFO (*ang. First-In, First-Out*, "pierwsze przyszło, pierwsze wyszło"), "typowa" kolejka; kolejność pobierania jest tożsama z kolejnością umieszczania elementów w kolejce).
   - LIFO (*ang. Last-In, First-Out*, "ostatnie przyszło, pierwsze wyszło").
2. Kolejki priorytetowe (o kolejności pobierania decyduje priorytet elementów kolejki; ale o tym będzie w późniejszym terminie).

Kolejki można przechowywać w listach wiązanych lub (w przypadku kolejek ograniczonych) w tablicach.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Jednym ze sposobów modelowania kolejek jest model "producenci i konsumenci":



Role producentów i konsumentów mogą pełnić np. procesy.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Operacje na kolejce:

- wstaw obiekt do kolejki,
- pobierz obiekt z czoła kolejki,
- usuń wszystkie elementy z kolejki,
- podaj rozmiar kolejki (aktualna liczbę elementów),
- sprawdź, czy kolejka jest pusta.

Operacje na kolejce:

- wstaw obiekt do kolejki,
- pobierz obiekt z czoła kolejki,
- usuń wszystkie elementy z kolejki,
- podaj rozmiar kolejki (aktualna liczbę elementów),
- sprawdź, czy kolejka jest pusta.

Przykład interfejsu kolejki w Javie:

```
package queues;
public interface Queue {
  public void enqueue(Object value);                          //wstaw do kolejki
  public Object dequeue() throws EmptyQueueException; //pobierz z kolejki
  public void clear();                                        //usuń wszystkie elementy
  public int size();                                          //podaj rozmiar kolejki
  public boolean isEmpty();                                   // true gdy kolejka jest pusta
}
```

Przykład implementacji kolejki FIFO na bazie listy wiązanej:

```
package queues;
import lists.LinkedList; // założenie, że w pakiecie lists są zdefiniowane
import lists.List;       // klasy LinkedList i List, omawiane wcześniej

public class EmptyQueueException extends RuntimeException {
}

public class ListFifoQueue implements Queue {
    private final List _list;

    public ListFifoQueue(List list) // konstruktor tworzący kolejkę na bazie
     {  _list = list; }             // istniejącej listy określonej jako argument

    public ListFifoQueue()          // konstruktor tworzący kolejkę na bazie
     { this(new LinkedList()); }    // utworzonej listy wiązanej
// c.d.n.
```

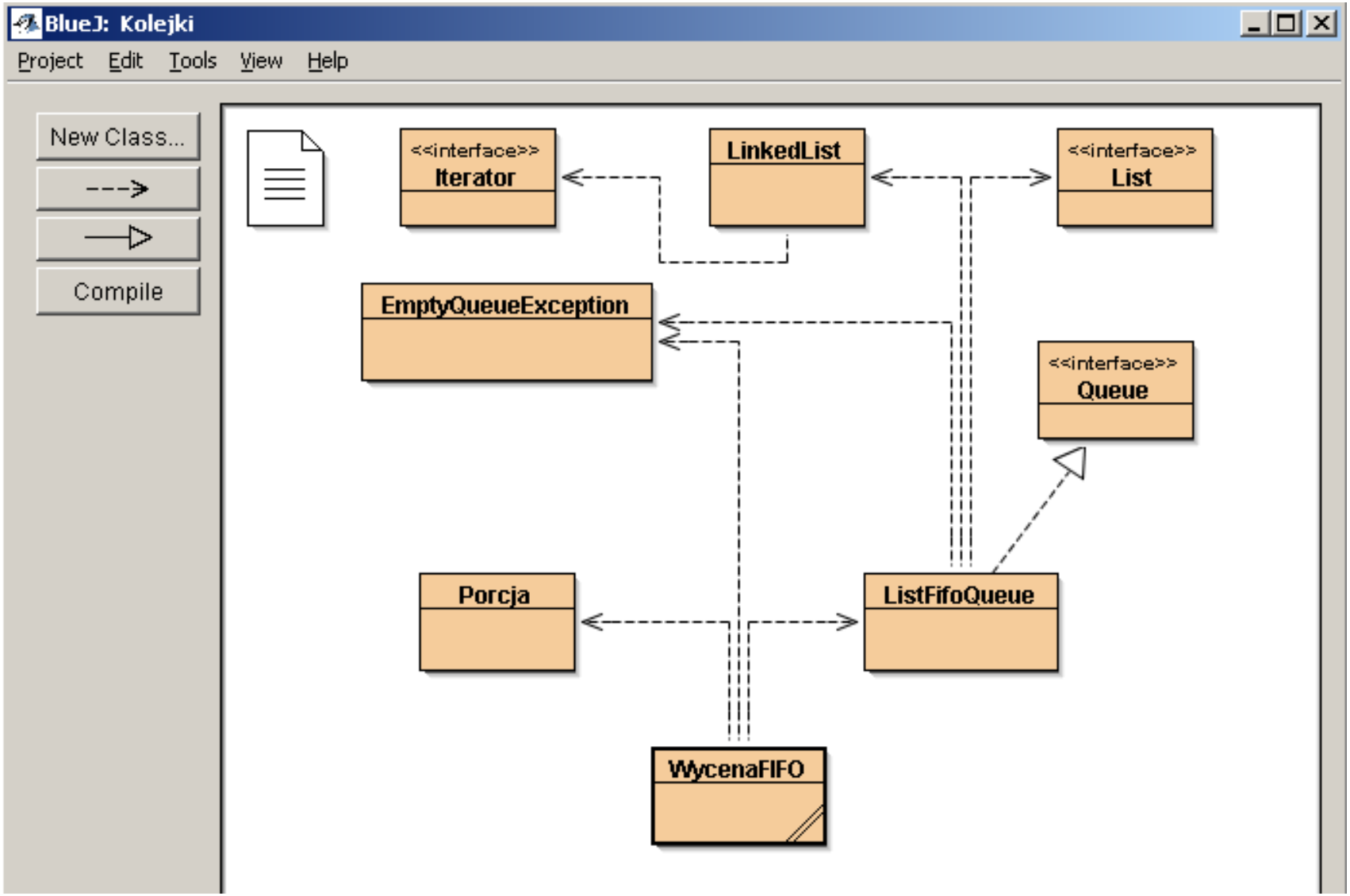
// Przykład implementacji kolejki FIFO na bazie listy wiązanej – c.d.

```
    public void enqueue(Object value) // wstawienie elementu do kolejki
     {  _list.add(value); }                    // to dopisanie go na końcu listy

    public Object dequeue() throws EmptyQueueException
     {  if (isEmpty())                                    // pobranie oraz usunięcie
          { throw new EmptyQueueException(); }    // elementu z czoła kolejki
        return _list.delete(0);                          // to usunięcie z listy
     }                    // elementu o indeksie 0, wraz ze zwrotem jego wartości
    // implementacje pozostałych metody interfejsu Queue są identyczne, jak
    // dla listy:
    public void clear() {  _list.clear(); }
    public int size()     {  return _list.size(); }
    public boolean isEmpty()  {  return _list.isEmpty(); }
    public String toString() { return _list.toString(); }
} // koniec przykładowej implementacji kolejki
```

Przykład kolejki – wycena rozchodów towarów w obrocie magazynowym

```
// operacja magazynowa opisuje liczbę sztuk oraz cenę jednostkową w zł:
public class Porcja {
  int szt;
  double cena;
  public Porcja(int liczbaSzt, double cenaJedn)
    {
       szt=liczbaSzt;
       cena=cenaJedn;
    }
  public int ilosc(){
      return szt;
    }
  public double cena(){
      return cena;
    }
// ....
}
```


BlueJ: Kolejki
Project Edit Tools View Help
New Class...
Compile
<<interface>>
Iterator
LinkedList
<<interface>>
List
EmptyQueueException
<<interface>>
Queue
Porcja
ListFifoQueue
WycenaFIFO

c.d.:

```
public class WycenaFIFO {
    int sumaPrzSzt=0,sumaRozSzt=0,sumaZapSzt=0;
    double wartPrz=0,wartRoz=0,wartZap=0;
    public WycenaFIFO() {}

    public ListFifoQueue przyjmijDoMagazynu(ListFifoQueue kolejka, String data,
                                                        Porcja p){
        int ilosc=p.ilosc();
        double cena=p.cena();
        for (int i=1; i<=ilosc; i++)
          kolejka.enqueue(new Porcja(1,cena));
        double wprz=ilosc*cena;
        sumaPrzSzt+=ilosc; wartPrz+=wprz;
        sumaZapSzt+=ilosc; wartZap+=wprz;
        System.out.printf("%10s %6.2f     %3d %8.2f                "+
                          " %3d %8.2f\n",
                          data,cena,ilosc,wprz,sumaZapSzt,wartZap);
        return kolejka;
    } // c.d.n.
```

```
// c.d.
public ListFifoQueue wydajZMagazynu(ListFifoQueue kolejka, String data, int lszt){
        Porcja p;
        System.out.printf("%10s                          %3d\n",data,lszt);
        double c=0, wrozch=0;
        int lc=0;
        try {   for (int i=1; i<=lszt; i++){
                  p=(Porcja) kolejka.dequeue();
                  if (c!=0 && p.cena!=c){
                       System.out.printf("      -->  %6.2f                  %3d    \n",c,lc);
                       lc=0; }
                    else c=p.cena();
                  lc++; c=p.cena();
                  wrozch+=c; sumaRozSzt++; wartRoz+=c; sumaZapSzt--; wartZap-=c;
                 }
                System.out.printf("      -->  %6.2f                  %3d %8.2f   %3d %8.2f\n",
                                        c,lc ,wrozch,sumaZapSzt,wartZap);
              }   catch (EmptyQueueException e) {System.out.println("      ???\n");}
        return kolejka;
    } // c.d.n.
```

```
public static void main(){
  System.out.println("\n    Ilustracja działania kolejki FIFO - Magazyn towarów");
  System.out.println("    Wycena rozchodów towarów w cenach przychodu (zakupu)");
  System.out.println("-------------------------------------------------------------------------");
  WycenaFIFO magazyn=new WycenaFIFO();
  System.out.println("   Data     Cena zł   - Przychód -  -- Rozchód --  --- Zapas ---");
  System.out.println(" operacji                 szt.   zł          szt.   zł          szt.   zł ");
  System.out.println("-------------------------------------------------------------------------");
  ListFifoQueue kolejka=new ListFifoQueue();
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-01",new Porcja(3,10.0));
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-03",new Porcja(4,15.0));
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-08",new Porcja(5,20.0));
  kolejka=magazyn.wydajZMagazynu(kolejka,"2008-04-09",6);
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-10",new Porcja(4,10.0));
  kolejka=magazyn.wydajZMagazynu(kolejka,"2008-04-10",8);
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-11",new Porcja(4,20.0));
  System.out.println("-------------------------------------------------------------");
  System.out.printf("Stan bieżący:        %3d %8.2f   %3d %8.2f   %3d %8.2f   \n",
     magazyn.sumaPrzSzt,magazyn.wartPrz,
     magazyn.sumaRozSzt,magazyn.wartRoz,
     magazyn.sumaZapSzt,magazyn.wartZap); }
} // koniec przykładu
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

BlueJ: Terminal Window - Kolejki

Options

```
    Ilustracja działania kolejki FIFO - Magazyn towarów
    Wycena rozchodów towarów w cenach przychodu (zakupu)
-------------------------------------------------------------------
   Data      Cena zł    - Przychód -   -- Rozchód --   --- Zapas ---
 operacji               szt.    zł      szt.    zł      szt.    zł
-------------------------------------------------------------------
2008-04-01  10,00         3     30,00                     3     30,00
2008-04-03  15,00         4     60,00                     7     90,00
2008-04-08  20,00         5    100,00                    12    190,00
2008-04-09                                6
      -->   10,00                        3
      -->   15,00                        3     75,00      6    115,00
2008-04-10  10,00         4     40,00                    10    155,00
2008-04-10                                8
      -->   15,00                        1
      -->   20,00                        5
      -->   10,00                        2    135,00      2     20,00
2008-04-11  20,00         4     80,00                     6    100,00
-------------------------------------------------------------------
Stan bieżący:            20    310,00     14    210,00      6    100,00
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wybrane przykłady zaawansowanego zastosowania kolejek:

1. W zarządzaniu zadaniami przez systemy operacyjne:
   algorytmy szeregowania zadań/procesów:
   - wybór najdłużej oczekującego procesu (FCFS – First Come First Serve),
   - wybór najkrócej wykonywanego procesu (SJF – Shortest Job First),
   - algorytm kolejki cyklicznej (Round-Robin), związany z kwantyfikacją czasu dla zadań; zadanie, które wyczerpało kwant czasu przechodzi na koniec kolejki a jako następne wykonywane jest kolejne zadanie (z czoła kolejki).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wybrane przykłady zaawansowanego zastosowania kolejek:

1. W zarządzaniu zadaniami przez systemy operacyjne:
   algorytmy szeregowania zadań/procesów:
   - wybór najdłużej oczekującego procesu (FCFS – First Come First Serve),
   - wybór najkrócej wykonywanego procesu (SJF – Shortest Job First),
   - algorytm kolejki cyklicznej (Round-Robin), związany z kwantyfikacją czasu dla zadań; zadanie, które wyczerpało kwant czasu przechodzi na koniec kolejki a jako następne wykonywane jest kolejne zadanie (z czoła kolejki).
2. Serializacja transakcji w rozproszonym przetwarzaniu danych.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wybrane przykłady zaawansowanego zastosowania kolejek:

1. W zarządzaniu zadaniami przez systemy operacyjne:
   algorytmy szeregowania zadań/procesów:
   - wybór najdłużej oczekującego procesu (FCFS – First Come First Serve),
   - wybór najkrócej wykonywanego procesu (SJF – Shortest Job First),
   - algorytm kolejki cyklicznej (Round-Robin), związany z kwantyfikacją czasu dla zadań; zadanie, które wyczerpało kwant czasu przechodzi na koniec kolejki a jako następne wykonywane jest kolejne zadanie (z czoła kolejki).
2. Serializacja transakcji w rozproszonym przetwarzaniu danych.
3. Kolejkowanie klientów w obsłudze przez serwer interaktywny (w architekturze klient-serwer).

Wybrane przykłady zaawansowanego zastosowania kolejek:

1. W zarządzaniu zadaniami przez systemy operacyjne:
   algorytmy szeregowania zadań/procesów:
   - wybór najdłużej oczekującego procesu (FCFS – First Come First Serve),
   - wybór najkrócej wykonywanego procesu (SJF – Shortest Job First),
   - algorytm kolejki cyklicznej (Round-Robin), związany z kwantyfikacją czasu dla zadań; zadanie, które wyczerpało kwant czasu przechodzi na koniec kolejki a jako następne wykonywane jest kolejne zadanie (z czoła kolejki).
2. Serializacja transakcji w rozproszonym przetwarzaniu danych.
3. Kolejkowanie klientów w obsłudze przez serwer interaktywny (w architekturze klient-serwer).
4. Zapewnienie synchronizacji dostępu do danych i procesów (kolejki blokujące).

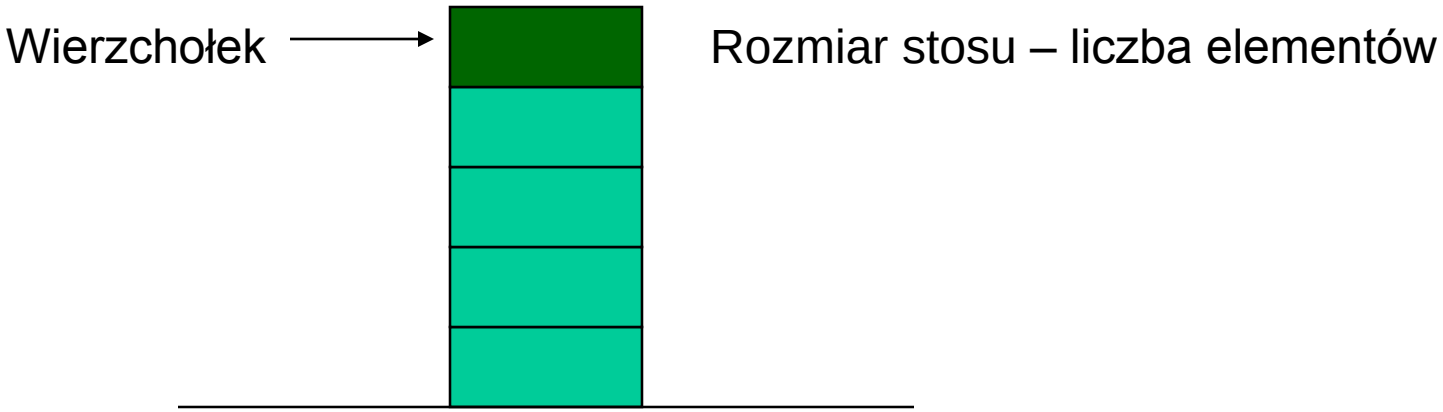
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wykład 5.
Stosy – implementacja, przetwarzanie

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Stosy

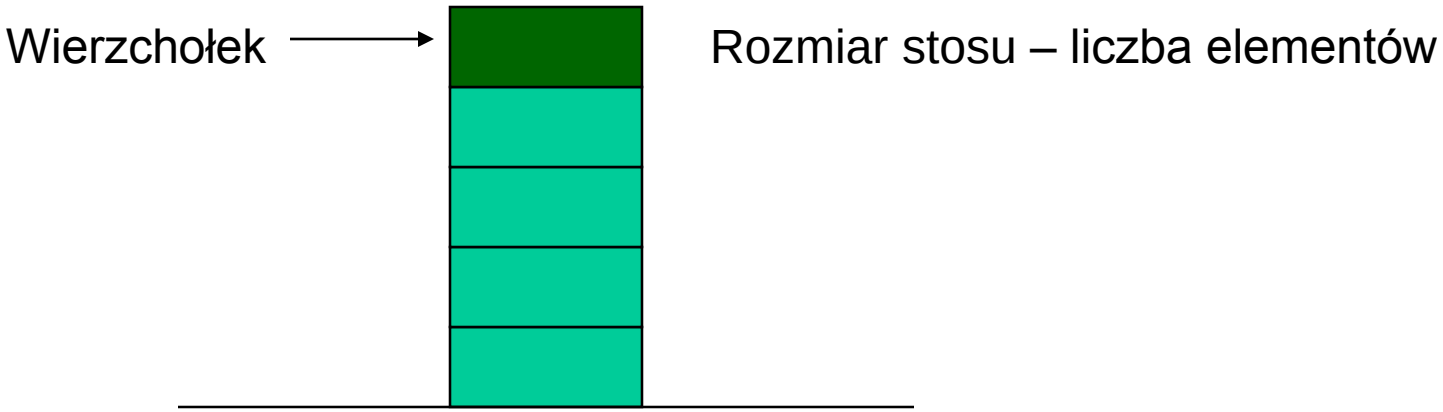
Stos (*ang. stack*) – struktura danych o charakterze kolejki LIFO (*Last-In, First-Out)*.



Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Stosy

Stos (*ang. stack*) – struktura danych o charakterze kolejki LIFO (*Last-In, First-Out)*.



Różnice w porównaniu do kolejek FIFO dotyczą dostępności elementów:

- w kolejkach operujemy na obu jej krańcach,
- w stosie dostępny jest tylko jeden kraniec – tzw. element szczytowy lub wierzchołkowy (*ang. top of stack),*
- jedynym dostępnym elementem jest ten, który przebywa na stosie najkrócej.

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),
- *pop* - zdjęcie i udostępnienie elementu z wierzchołka stosu (rozmiar stosu zmniejsza się o 1); próba zdjęcia elementu z pustego stosu powoduje wystąpienie wyjątku EmptyStackException,

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),
- *pop* - zdjęcie i udostępnienie elementu z wierzchołka stosu (rozmiar stosu zmniejsza się o 1); próba zdjęcia elementu z pustego stosu powoduje wystąpienie wyjątku EmptyStackException,
- *size* – zwraca rozmiar stosu (liczbę elementów znajdujących się na stosie,

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),
- *pop* - zdjęcie i udostępnienie elementu z wierzchołka stosu (rozmiar stosu zmniejsza się o 1); próba zdjęcia elementu z pustego stosu powoduje wystąpienie wyjątku EmptyStackException,
- *size* – zwraca rozmiar stosu (liczbę elementów znajdujących się na stosie,
- *peek* – zwraca wartość elementu szczytowego (wierzchołka) bez zdejmowania go ze stosu; próba wykonania tej operacji na pustym stosie powoduje wystąpienie wyjątku EmptyStackException,

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),
- *pop* - zdjęcie i udostępnienie elementu z wierzchołka stosu (rozmiar stosu zmniejsza się o 1); próba zdjęcia elementu z pustego stosu powoduje wystąpienie wyjątku EmptyStackException,
- *size* – zwraca rozmiar stosu (liczbę elementów znajdujących się na stosie,
- *peek* – zwraca wartość elementu szczytowego (wierzchołka) bez zdejmowania go ze stosu; próba wykonania tej operacji na pustym stosie powoduje wystąpienie wyjątku EmptyStackException,
- *isEmpty* – sprawdzenie, czy stos jest pusty (czyli: czy size()=0),

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),
- *pop* - zdjęcie i udostępnienie elementu z wierzchołka stosu (rozmiar stosu zmniejsza się o 1); próba zdjęcia elementu z pustego stosu powoduje wystąpienie wyjątku EmptyStackException,
- *size* – zwraca rozmiar stosu (liczbę elementów znajdujących się na stosie,
- *peek* – zwraca wartość elementu szczytowego (wierzchołka) bez zdejmowania go ze stosu; próba wykonania tej operacji na pustym stosie powoduje wystąpienie wyjątku EmptyStackException,
- *isEmpty* – sprawdzenie, czy stos jest pusty (czyli: czy size()=0),
- *clear* – usunięcie wszystkich elementów ze stosu (stos staje się pusty).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Stosy

Podstawowe operacje na stosie:

- *push* – odłożenie (*położenie*) elementu na stos (rozmiar stosu zwiększa się o 1),
- *pop* - zdjęcie i udostępnienie elementu z wierzchołka stosu (rozmiar stosu zmniejsza się o 1); próba zdjęcia elementu z pustego stosu powoduje wystąpienie wyjątku EmptyStackException,
- *size* – zwraca rozmiar stosu (liczbę elementów znajdujących się na stosie,
- *peek* – zwraca wartość elementu szczytowego (wierzchołka) bez zdejmowania go ze stosu; próba wykonania tej operacji na pustym stosie powoduje wystąpienie wyjątku EmptyStackException,
- *isEmpty* – sprawdzenie, czy stos jest pusty (czyli: czy size()=0),
- *clear* – usunięcie wszystkich elementów ze stosu (stos staje się pusty).

Stosy (tak, jak kolejki) mogą być ograniczone lub nieograniczone.

Stosy

Stos realizuje następujący interfejs:

```
package stacks;
import queues.Queue;

public interface Stack extends Queue {
    public void push(Object value); // odłóż na stos
    public Object pop() throws EmptyStackException; //pobierz ze stosu
    public Object peek() throws EmptyStackException; //odczytaj ze stosu,
                                                     // pozostaw element na stosie
    public void clear();      // usuń wszystkie elementy ze stosu (czyść stos)
    public int size();                // daj rozmiar stosu ("wysokość" stosu)
    public boolean isEmpty(); // true jeśli stos jest pusty
}
// Należy zadeklarować klasę wyjątku EmptyStackException:
public class EmptyStackException extends RuntimeException {}
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykładowa implementacja stosu z wykorzystaniem listy wiązanej i wskazaniem, że stos jest rodzajem kolejki.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykładowa implementacja stosu z wykorzystaniem listy wiązanej i wskazaniem, że stos jest rodzajem kolejki.

Uwaga: Stos (szczególnie ograniczony) można efektywnie zaimplementować przy użyciu tablicy. Wówczas można zastąpić bezpośrednie tworzenie listy wiązanej (jak w przykładzie poniżej) odpowiednim konstruktorem, pozwalającym wybierać rodzaj implementacji stosu (tablica lub lista wiązana). -> Proszę to zrobić samodzielnie.

Przykładowa implementacja stosu z wykorzystaniem listy wiązanej i wskazaniem, że stos jest rodzajem kolejki.

Uwaga: Stos (szczególnie ograniczony) można efektywnie zaimplementować przy użyciu tablicy. Wówczas można zastąpić bezpośrednie tworzenie listy wiązanej (jak w przykładzie poniżej) odpowiednim konstruktorem, pozwalającym wybierać rodzaj implementacji stosu (tablica lub lista wiązana). -> Proszę to zrobić samodzielnie.

```
package stacks;
import lists.LinkedList;
import lists.List;
import queues.EmptyQueueException;

public class ListStack implements Stack {
    private final List _list = new LinkedList();
    public void push(Object value) { _list.add(value); }
    public Object pop() throws EmptyStackException {
        if (isEmpty()) { throw new EmptyStackException(); }
        return _list.delete(_list.size() - 1);
    }
// c.d.n.
```

```
// c.d.
public Object peek() throws EmptyStackException {
    Object result = pop();
    push(result);
    return result;
}
public void enqueue(Object value)  { push(value); }

public Object dequeue() throws EmptyQueueException {
   try {
      return pop();
   }  catch (EmptyStackException e) { throw new EmptyQueueException(); }
}
public void clear()  { _list.clear(); }

public int size()      { return _list.size(); }

public boolean isEmpty() { return _list.isEmpty(); }
} // koniec przykładowej implementacji stosu
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykładami zastosowania stosu są:

1. kalkulator obliczający wartości wyrażeń w ONP (odwrotnej notacji polskiej, tzw. postfiksowej; wyrażenie: a + b ma postać: a b +)
   ONP pozwala zapisać wyrażenie bez użycia nawiasów, gdyż jednoznacznie określa kolejność wykonywania działań.

   Algorytm obliczenia wartości wyrażenia ONP z użyciem stosu:

   - Wyczyść stos.
   - Dla wszystkich symboli z wyrażenia ONP wykonuj:
     - jeśli i-ty symbol jest liczbą, to odłóż go na stos,
     - jeśli i-ty symbol jest operatorem to:
       - zdejmij ze stosu jeden element (ozn. *a*),
       - zdejmij ze stosu kolejny element (ozn. *b*),
       - odłóż na stos wartość *b operator a*.
     - jeśli i-ty symbol jest funkcją to:
       - zdejmij ze stosu oczekiwaną liczbę parametrów funkcji (ozn. $a_1...a_n$)
       - odłóż na stos wynik funkcji dla parametrów $a_1...a_n$
   - Zdejmij ze stosu wynik.

2. Zarządzanie ekranami aplikacji

("rozwijanie/otwieranie" i "zwijanie/zamykanie" kolejnych ekranów),

2. Zarządzanie ekranami aplikacji
   ("rozwijanie/otwieranie" i "zwijanie/zamykanie" kolejnych ekranów),

3. Zarządzanie ciągiem zmian (np. w edytorze tekstu), z możliwością ich wycofywania i ponawiania (*undo* i *redo*).

2. Zarządzanie ekranami aplikacji
   ("rozwijanie/otwieranie" i "zwijanie/zamykanie" kolejnych ekranów),

3. Zarządzanie ciągiem zmian (np. w edytorze tekstu), z możliwością ich wycofywania i ponawiania (*undo* i *redo*).

4. Odwracanie kolejności wyrazów ciągu.

2. Zarządzanie ekranami aplikacji
   ("rozwijanie/otwieranie" i "zwijanie/zamykanie" kolejnych ekranów),

3. Zarządzanie ciągiem zmian (np. w edytorze tekstu), z możliwością ich wycofywania i ponawiania (*undo* i *redo*).

4. Odwracanie kolejności wyrazów ciągu.

5. Różne algorytmy kombinatoryczne i symulacyjne.

2. Zarządzanie ekranami aplikacji
("rozwijanie/otwieranie" i "zwijanie/zamykanie" kolejnych ekranów),

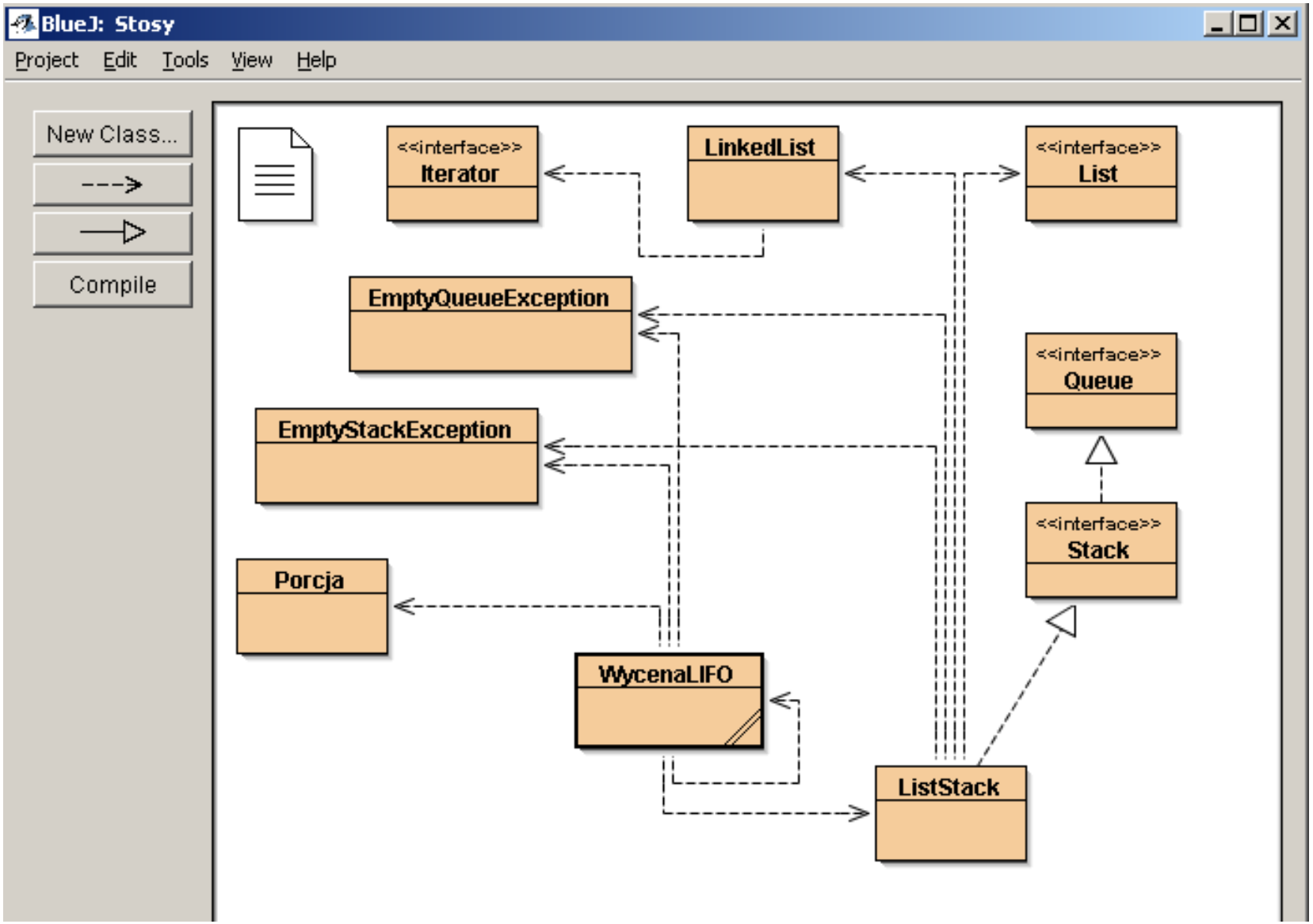
3. Zarządzanie ciągiem zmian (np. w edytorze tekstu), z możliwością ich wycofywania i ponawiania (*undo* i *redo*).

4. Odwracanie kolejności wyrazów ciągu.

5. Różne algorytmy kombinatoryczne i symulacyjne.

W załączonym (w dalszej części) przykładzie zastosowano stos do wyceny rozchodów towarów w obrocie magazynowym według reguły LIFO (proszę porównać wyniki z wynikami uzyskanymi w przykładzie z kolejką FIFO).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar



```
// Przykład użycia stosu do wyceny rozchodów towarów według reguły LIFO:
public class WycenaLIFO {
    int sumaPrzSzt=0,sumaRozSzt=0,sumaZapSzt=0;
    double wartPrz=0,wartRoz=0,wartZap=0;
    public WycenaLIFO() {}

    public ListStack przyjmijDoMagazynu(ListStack kolejka, String data, Porcja p){
        assert p.ilosc()<=0:"Brak przychodu";
        int ilosc=p.ilosc();
        double cena=p.cena();
        for (int i=1; i<=ilosc; i++)
          kolejka.push(new Porcja(1,cena)); // kolejka.enqueue(new Porcja(1,cena));
        double wprz=ilosc*cena;
        sumaPrzSzt+=ilosc; wartPrz+=wprz;
        sumaZapSzt+=ilosc; wartZap+=wprz;
        System.out.printf("%10s %6.2f     %3d %8.2f                "+
                          " %3d %8.2f\n",
                          data,cena,ilosc,wprz,sumaZapSzt,wartZap);
        return kolejka;
    }
// c.d.n.
```

```
// c.d.
 public ListStack wydajZMagazynu(ListStack kolejka, String data, int lszt){
       assert lszt<=0:"Brak rozchodu";
       Porcja p;
       System.out.printf("%10s                          %3d\n",data,lszt);
       double c=0,wrozch=0;
       int lc=0;
       try {
       for (int i=1; i<=lszt; i++){
         p=(Porcja) kolejka.pop(); //p=(Porcja) kolejka.dequeue();
         if (c!=0 && p.cena!=c){
            System.out.printf("      -->  %6.2f                  %3d    \n",c,lc);
            lc=0;
          }
          else c=p.cena(); lc++; c=p.cena();
         wrozch+=c; sumaRozSzt++; wartRoz+=c; sumaZapSzt--; wartZap-=c;
       }
       System.out.printf("      -->  %6.2f                  %3d %8.2f   %3d %8.2f\n",
                         c,lc ,wrozch,sumaZapSzt,wartZap);
          }   catch (EmptyStackException e) {System.out.println("      ???\n");}
       return kolejka;
   }  // c.d.n.
```

```
// c.d.
public static void main(){
  System.out.println("\n    Ilustracja działania kolejki LIFO - Magazyn towarów");
  System.out.println("    Wycena rozchodów towarów w cenach przychodu (zakupu)");
  System.out.println("-------------------------------------------------------------------------------");
  WycenaLIFO magazyn=new WycenaLIFO();
  System.out.println("   Data     Cena zł   - Przychód -  -- Rozchód --  --- Zapas ---");
  System.out.println(" operacji            szt.   zł      szt.   zł      szt.   zł ");
  System.out.println("-----------------------------------------------------------------");
  ListStack kolejka=new ListStack();
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-01",new Porcja(3,10.0));
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-03",new Porcja(4,15.0));
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-08",new Porcja(5,20.0));
  kolejka=magazyn.wydajZMagazynu(kolejka,"2008-04-09",6);
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-10",new Porcja(4,10.0));
  kolejka=magazyn.wydajZMagazynu(kolejka,"2008-04-10",8);
  kolejka=magazyn.przyjmijDoMagazynu(kolejka,"2008-04-11",new Porcja(4,20.0));
  System.out.println("-----------------------------------------------------------------");
  System.out.printf("Stan bieżący:        %3d %8.2f   %3d %8.2f   %3d %8.2f   \n",
                          magazyn.sumaPrzSzt,magazyn.wartPrz,
                          magazyn.sumaRozSzt,magazyn.wartRoz,
                         magazyn.sumaZapSzt,magazyn.wartZap); }
} // koniec przykładu
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

BlueJ: Terminal Window - Stosy

Options

```
    Ilustracja działania kolejki LIFO - Magazyn towarów
    Wycena rozchodów towarów w cenach przychodu (zakupu)
-------------------------------------------------------------------
   Data      Cena zł    - Przychód -   -- Rozchód --   --- Zapas ---
 operacji               szt.    zł      szt.    zł      szt.    zł
-------------------------------------------------------------------
2008-04-01  10,00         3     30,00                     3     30,00
2008-04-03  15,00         4     60,00                     7     90,00
2008-04-08  20,00         5    100,00                    12    190,00
2008-04-09                              6
      -->   20,00                      5
      -->   15,00                      1    115,00       6     75,00
2008-04-10  10,00         4     40,00                    10    115,00
2008-04-10                              8
      -->   10,00                      4
      -->   15,00                      3
      -->   10,00                      1     95,00       2     20,00
2008-04-11  20,00         4     80,00                     6    100,00
-------------------------------------------------------------------
Stan bieżący:            20    310,00     14    210,00      6    100,00
```