Składnia i podstawy języka

- opisać wszystkie pętle w javie

-for
-while
-do...while
-for each - pętla for each to konstrukcja, która pozwala na sekwencyjne przeglądanie różnych zbiorów danych. Mogą nimi być tablice, a także dynamiczne struktury jak na przykład listy. W tym drugim przypadku musimy jednak skorzystać z Iteratora, a ogólniej mówiąc obiekty typu Iterable.

- wszystko o static

Modyfikator "static" używany jest, aby stworzyć zmienne, czy metody, dostępne dla każdej instancji klasy. Taka zmienna lub metoda istnieje zawsze, nawet gdy nie nie została utworzona żadna instancja klasy.
Statycznymi oznaczyć można:
metody,
zmienne,
klasy zagnieżdżone w innej klasie, ale nie wewnątrz metody,
bloki inicjalizacyjne.

Statycznymi nie można oznaczyć:
-konstruktorów,
-nie zagnieżdżonych klas,
-interfejsów,
-lokalnych zmiennych,
-metod wewnętrznych (zagnieżdżonych) klas
-zmiennych instancji.
Aby odwołać się do zmiennej statycznej można użyć obiektu tej klasy, lub nazwy klasy: (NazwaKlasy.poleStatyczne; NazwaKlasy.metodaStatyczna;)
W metodach statycznych, nie można odwoływać do zmiennych instancji (nie statycznych).
Metody statycznie nie mogą być przeciążane w klasie potomnej, mogą być co najwyżej redefiniowane.

INNA ODPOWIEDZ

Atrybuty i metody statyczne są związane z klasą, a nie z jej instancjami. Są więc dostępne dla każdej instancji klasy. Metody statyczne nie mogą więc bezpośrednio wywoływać zwykłych metod lub korzystać ze zwykłych atrybutów, gdyż są one określone jedynie w kontekście obiektów.
Atrybuty statyczne są inicjalizowane bezpośrednio po załadowaniu klasy przez JVM. Atrybuty zwykłe (niestatyczne) są inicjowane w momencie utworzenia obiektu (wywołania konstruktora). Zadeklarowane, a niezainicjowane atrybuty są ustawiane na 0 lub null.
Statycznymi oznaczyć można: metody, zmienne, klasy zagnieżdżone w innej klasie, ale nie wewnątrz metody, bloki inicjalizacyjne.
Statycznymi nie można oznaczyć: konstruktorów, niezagnieżdżonych klas, interfejsów, lokalnych zmiennych, metod zagnieżdżonych klas i zmiennych instancji.
Aby odwołąć się do zmiennej statycznej można użyć obiektu tej klasy lub nazwy klasy. (NazwaKlasy.poleStatyczne; NazwaKlasy.metodaStatyczna;)

- Final

Po klasie oznaczonej jako “final” nie można dziedziczyć.
Metoda oznaczona słowem “final” nie może zostać nadpisana w sub-klasie (metody abstrakcyjne nie mogą zostać oznaczone jako “final” (ani jako private, czy też static), gdyż tworzone są właśnie po to, aby nadpisać je, w sub-klasach).
Zmienną oznaczoną słowem “final”, raz zainicjalizowana, nie może zostać już zmieniona. Można zadeklarować zmienną jako final, bez inicjalizowania jej, wtedy pierwsza inicjalizacja będzie możliwa, kolejne już nie.
Referencja do obiektu oznaczona słowem “final” nie może zostać zmieniona (możliwa jest zmiana wszystkich pól obiektu, ale nie można już zmienić referencji tak, aby pokazywała na inny obiekt, warto zaznaczyć również, iż nie ma finalnych obiektów, jedynie referencje do nich mogą być finalne).
Parametry metody oznaczone jako finalne (ostateczne), nie mogą zostać zmienione w ciele metody.
Synchronized
Specyfikator synchronized daje nam gwarancję, że metoda w ten sposób oznaczona będzie mogła być wywołana przez jeden wątek w danym czasie (wątki zostaną opisane w późniejszym czasie).
Specyfikator synchronized może być wykorzystany tylko dla metod, może być również użyty z “public”, “protected”, “private”, albo default (zasięg pakietowy, bez dodatkowego specyfikatora).

- co to jest interfejs?

Interfejs to szkielet, albo bardziej szczegółowo wymóg jaki będą musiały spełniać klasy implementujące nasz interfejs.
Interfejsy zostały stworzone, dla klas o podobnych właściwościach, lub zachowaniach, ale nie powiązanych ze sobą (dziedziczenie), posiadają za to pewne wspólne cechy. lub wymagane jest, aby je posiadały. Każda klasa, która implementuje interfejs musi implementować wszystkie jego metody, nie ma wyjątków.
W przeciwieństwie do dziedziczenia, interfejsów z którymi powiązana jest dana klasa może być więcej niż jeden.
Przykładowa metoda interfejsu wygląda następująco:
abstract void getSlideFactor();
Zauważyć należy, że metoda zakończona jest średnikiem, a nie parą nawiasów klamrowych.
Właściwości interfejsów:
Wszystkie metody interfejsu są domyślnie publiczne i abstrakcyjne, nie trzeba tego zaznaczać przy deklaracji.
wszystkie pola interfejsu są domyślnie publiczne, statyczne i finalne.
Metody interfejsów nie mogą być statyczne, nie mogą również być oznaczone jako final (finalnych metod nie można nadpisać w klasie implementującej interfejs).
Interfejsy mogą rozszerzać jedynie inne interfejsy - (pierwsza nieabstrakcyjna klasa, która będzie implementować interfejs rozszerzający inny interfejs, będzie musiała zaimplementować wszystkie metody z obu interfejsów).
Interfejs nie może implementować innego interfejsu.
Deklaracja interfejsu odbywa się przy pomocy słowa kluczowego “interface”

INNA ODPOWIEDZ

Interfejs jest typem abstrakcyjnym zawierającym jedynie deklaracje metod oraz stałych (zmiennych oznaczonych jako 'static' i 'final'). Nie można tworzyć instancji interfejsów. Są one implementowane (słowo kluczowe 'implements') przez klasy, które muszą zawierać definicje

wszystkich metod zawartych w interfejsie. Możliwa jest jednoczesna implementacja wielu interfejsów. Wszystkie metody w interfejsie są abstrakcyjne więc zbędne jest używanie słowa 'abstract'.

- interfejs a klasa abstrakcyjna?

O interfejsach należy myśleć jako o całkowicie abstrakcyjnych klasach. Interfejs może mieć metody, ale tylko abstrakcyjne, czyli nie może zawierać ich implementacji. Dla przypomnienia, klasy abstrakcyjne mogą zawierać zarówno metody abstrakcyjne (bez implementacji), jak i zwykłe metody (z implementacją w klasie abstrakcyjnej). Jaka jest więc przewaga klas abstrakcyjnych nad interfejsami, skoro na pierwszy rzut oka nie widać praktycznie żadnych różnic? Przede wszystkim tutaj możemy już umieścić jakąś implementację, natomiast klasy rozszerzające będą miały jedynie obowiązek rozbudować funkcjonalność poprzez zaimplementowanie metod abstrakcyjnych.
W przypadku, gdy przykładowo chcemy w 4 klasach uzyskać 4 metody o identycznej funkcjonalności, różniące się implementacją tylko jednej nie ma zbytnio sensu tworzyć najpierw interfejsu, później klasy implementującej 4 metody Niewątpliwą wadą będzie natomiast to, że w Javie możemy implementować wiele interfejsów, ale rozszerzać (extend) tylko jedną klasę.

INNA ODPOWIEDZ

Można implementować wiele interfejsów lecz dziedziczyć tylko po jednej klasie.
Wszystkie zadeklarowane metody w interfejsie są publiczne, natomiast metody zadeklarowane w klasach mogą być publiczne, prywatne lub chronione.
Interfejs może zawierać jedynie deklaracje metod, a klasa abstrakcyjna może zawierać również metody zdefiniowane.
Klasy abstrakcyjne mogą zawierać atrybuty, natomiast interfejsy nie (w nich wszystkie atrybuty są domyślnie public, static i final).

- co to jest klasa abstrakcyjna?

Klasa abstrakcyjna to klasa, której jedna lub więcej metod jest abstrakcyjna (klasa ta zawiera jedynie deklarację takiej metody, która oznaczona jest słowem 'abstract'. np. public abstract int doSomething()), nie można bezpośrednio tworzyć instancji klasy abstrakcyjnej. Oznaczana jest słowem kluczowym 'abstract' (np. public abstract class AnyClass). Klasa może mieć tylko jednego bezpośredniego rodzica. Dziedzicząc używamy słowa 'extends' (np. public class AnyClass extends AnotherClass...). Jeśli klasa nie posiada rodzica, dziedziczy automatycznie po klasie Object (java.lang.Object) W związku z tym instancja dowolnej klasy jest obiektem (instancją klasy Object).

- Opisać intrukcje warunkowe w javie (if, else, switch)

- Pytanie o argumenty switch

(typy primitywne + od 1.7 także obiekty string)
Pytanie o argumenty switch
Instrukcji wyboru SWITCH używamy, jeśli chcemy w zależności od wartości pewnego wyrażenia wykonać jeden z kilku fragmentów kodu.

W instrukcji SWITCH – inaczej niż w instrukcji IF – nie określamy warunku, który musi być prawdziwy, aby ujęty w instrukcji kod się wykonał. Tutaj określamy {wyrażenie wyboru}, którego wartością jest liczba całkowita i definiujemy warianty, które mają się wykonać w zależności od tego, jaka jest wartość tego wyrażenia. Warianty oznaczamy słówkiem kluczowym case, za którym umieszczamy wartość dla której kod z tego wariantu ma być uruchomiony. Dodatkowo, instrukcja SWITCH umożliwia określenie wariantu domyślnego, który to wariant oznaczamy słówkiem kluczowym default. Wariant ten wykonywany jest, jeśli dla danej wartości {wyrażenia wyboru} nie określono wariantu typu case.
Co ważne, wariant jest tylko punktem początkowym i kod będzie się wykonywał przechodząc kaskadowo przez kolejne warianty, aż do końca instrukcji SWITCH, chyba że wykonanie to przerwiemy za pomocą instrukcji break
(typy primitywne + od 1.7 także obiekty string)
Istnieją dwa rodzaje typów: typy obiektowe i typy proste (prymitywne). Typy obiektowe definiowane są przez klasy – i jest bez znaczenia, czy są to klasy które sami zaimplementowaliśmy, czy klasy pochodzące z jakichś bibliotek, np. bibliotek Java SE. Typy proste to typy wbudowane w język, traktowane w specyficzny sposób. Jest 8 typów prostych; są to:char, boolean, byte, short, int, long, float oraz double.
Typ char reprezentuje pojedynczy znak (np. literę). Zmienne typu char mają wartości odpowiadające dowolnemu znakowi kodowania UTF-16.
Typ boolean reprezentuje wartość typu logicznego. Zmienna typu boolean może przyjmować jedną z dwu wartości: true albo false.
Pozostałe typy proste to typy numeryczne. Typy: byte, short, int i long to typy całkowitoliczbowe tj. typy reprezentujące liczby całkowite.
Typy float oraz double reprezentują liczby zmiennoprzecinkowe. Typ float reprezentuje wartości pojedynczej precyzji a typ double liczby podwójnej precyzji.
Bodaj najpopularniejszym typem obiektowym (pochodzącym z biblioteki Java SE) jest typ String. Jest to typ reprezentujący ciąg znaków.

- pakiety

Program w języku Java to – w pewnym uproszczeniu – zbiór klas. Nie jest to jednak zbiór bezładny. Klasy pogrupowane są w tzw. pakiety. Klasy dzielimy na pakiety by pogrupować je według ich znaczenia – analogicznie do tego, jak dzielimy pliki na katalogi na dysku twardym naszego komputera.
Pakiety – podobnie jak katalogi – mają strukturę hierarchiczną, tj. każdy z pakietów może zawierać kolejne pakiety – podobnie jak katalogi mogą zawierać podkatalogi. Każdy pakiet, oprócz dowolnej liczby innych pakietów może zawierać także dowolną liczbę klas. O klasach które nie należą do żadnego pakietu (mówi się o nich czasem, że należą do pakietu domyślnego lub głównego) możemy myśleć jak o plikach które znajdują się bezpośrednio na dysku C naszego komputera – one również nie należą do żadnego katalogu.
Nazwy pakietów zwyczajowo pisze się małymi literami. Kolejne poziomy zagnieżdżenia w hierarchii pakietów oddziela się od siebie kropkami, podobnie do tego jak nazwy kolejnych katalogów w ścieżce dostępu oddziela się od siebie ukośnikami. Plik zawierający implementację tej klasy musi znajdować się w katalogu odpowiadającym temu pakietowi.
Implementując klasę musimy określić w jakim znajduje się ona pakiecie. W tym celu, na samym początku pliku umieszczamy instrukcję:
package {nazwa pakietu};

INNA ODPOWIEDZ

Pakiety są hierarchicznymi zbiorami klas. Pozwalają na grupowanie, systematyzowanie i katalogowanie klas w znacznym stopniu ułatwiając korzystanie z języka. Nazwa pakietu, do którego należy klasa jest podana w pliku definiującym klasę:

package pakiet.podpakiet;
Jeśli chcemy użyć klasy z innego pakietu niż nasz, musimy ją uprzednio zaimportować:
import pakiet.podpakiet.Klasa;
lub jeśli chcemy zaimportować wszystkie klasy znajdujące się w danym pakiecie/podpakiecie:
improt pakiet.podpakiet.*;
Hierarchia pakietów jest odwzorowana w systemie plików w hierarchię katalogów. Konwencja nazewnicza pakietów sugeruje stosowanie odwrotnych nazw domenowych. Np.
pl.uj.edu.fais.java.wyklad2

- Czy można utworzyć obiekt klasy abstrakcyjnej i kilka szczegółów na ten temat.

(nie można, dziedziczenie)
Ich użycie, a także budowa jest bardzo podobna do interfejsów, ale zawiera też pewne różnice, poniżej znajduje się lista najważniejszych cech klas abstrakcyjnych:

mogą zawierać metody abstrakcyjne, czyli takie, które nie posiadają implementacji (ani nawet nawiasów klamrowych)

może zawierać stałe (zmienne oznaczone jako public static final)

mogą zawierać zwykłe metody, które niosą jakąś funkcjonalność, a klasy rozszerzające mogą ją bez problemu dziedziczyć

klasy rozszerzające klasę abstrakcyjną muszą stworzyć implementację dla metod oznaczonych jako abstrakcyjne w klasie abstrakcyjnej

metod abstrakcyjnych nie można oznaczać jako statyczne (nie posiadają implementacji)

podobnie jak w przypadku interfejsów nie da się tworzyć instancji klas abstrakcyjnych

- Jak skopiowac obiekt w Javie - metoda clone(), dlaczego np. Vector ma własna implementacje clonnalble

Vector v1, v2;
v1 = new Vector();
v2 = v1;
Tutaj v1 i v2 to REFERENCJE do tego samego obiektu.
Aby skopiować obiekt:
v2 = (Vector)v1.clone();
Teraz v1 i v2 wskazują na różne bliźniacze obiekty.
Implementacje interfejsu Clonnable informuje, że nasz obiekt wspiera klonowanie. Metoda clone() jest zaimplementowana w klasie Object.

inna odpowiedz

W Javie obiekty są obsługiwane za pomocą referencji i nie ma operatora dla kopiowania obiektu.
Operator przypisania duplikuje referencję a nie sam obiekt. Metoda clone() zapewnia brakującą funkcjonalność. Implementacja interfejsu Cloneable informuje, że nasz obiekt wspierea

klonowanie. Bazowa metoda clone jest zaimplementowana w klasie Object. Dzięki niej dochodzi do faktycznego skopiowania (sklonowania) obiektu.
Przykładowe wywołanie:
Vector v1, v2;
v1 = new Vector();
v2 = v1; // v2 i v1 wskazują na ten sam obiekt
v2 = (Vector) v1.clone(); //tworzona jest kopia obiektu, v1 i v2 wskazują na dwa różne, bliźniacze obiekty.
Klasa Vector implementuje własny sposób klonowania ze względu na to iż domyślna definicja funkcji clone() jest określana jako 'płytka' i w przypadku typów tak złożonych jak Vector niewystarczająca. Przy użyciu płytkiego klonowania pola oryginału, które oznaczają referencje do obiektów, kopiowane są jako referencje (nie ma przy tym tworzenia kopii obiektów na które te referencje wskazują). Często takie zachowanie jest nieodpowiednie (bo kopia nie będzie niezależna, będzie miała wspólne części danych z oryginałem). Dlatego metodę clone() można przedefiniować tak jak to dzieje się w przypadku rozpatrywanego typu danych. Dzięki temu osiągamy 'głęboką kopię' obiektu.

- Javadoc - co to jest, jak działa, napisac program z komentarzami ważne /**komentarz */

Narzędzie to generuje dokumentację kodu źródłowegoJavy na podstawie zamieszczonych w kodzie komentarzy Javadoc.
Komentarz Javadoc oddzielony jest znacznikami /** i */, które sprawiają, że ich zawartość (czyli to, co znajduje się między nimi), jest ignorowana przez kompilator.
Pierwsza jego linia to opis metody lub klasy, która zadeklarowana jest poniżej. Dalej znajduje się pewna liczba opcjonalnych tagów, które z kolei opisują parametry metody (@param), wartość zwracaną (@return) itp.

Tagi używamy w odpowiedniej kolejności:
@author (classes and interfaces only, required)
@version (classes and interfaces only, required)
@param (methods and constructors only)
@return (methods only)
@exception
@see
@since
@serial
@deprecated

INNA ODPOWIEDZ

Jest generatorem dokumentacji znacząco ułatwiającym komentowanie pisanego kodu. Narzędzie to generuje dokumentację kodu źródłowego Javy na podstawie zamieszczonych w kodzie komentarzy Javadoc. Komentarz Javadoc jest oddzielony znacznikami /** i */ Dalej znajduje się pewna liczba opcjonalnych tagów opisujących min parametry metody (@param), autora programu (@author), zwracane wyjątki (@throws) czy wartość zwracaną (@return).
Generowanie dokumentacji: javadoc TryAndCheck.java

- print, printf - roznice

Znasz dwie metody z System.out – print i println. Istnieje również kolejna: printf. “F” jest skrótem od formatowania. Metoda printf różni się w pewnym stopniu od tych dwóch pozostałych. Posiada ona kilka argumentów. Ściślej – nieskończenie wiele. Pierwszym argumentem jest tekst, który chcesz wyświetlić, a kolejnymi (opcjonalnymi) są argumenty dla tzw. specyfikatorów formatu.
System.out.printf("Jakiś tekst", argument1, argument2, argument3...);
Kiedy w metoda printf trafia w łańcuchu na znak procentu, od razu zaczyna analizować jaki to specyfikator formatu.
%d – liczba dziesiętna, całkowita (za przykład użyliśmy typu int),
%f – liczba zmiennoprzecinkowa (w naszym wypadku: float),
%s – łańcuch tekstowy,
%c – znak,
%b – wartość logiczna (czyli typu boolean – true lub false).
%n – jest tym samym, co \n w print lub w println.
Flagi metody printf - jest to standardowy sposób w jaki instrukcja wyświetla liczby zmiennoprzecinkowe. Metoda jednak jest na tyle elastyczna, że można format dowolnie zmieniać. Służą do tego flagi. Flagi, to znaczniki, które modyfikują sposób wyświetlania danego specyfikatora formatu.
Znaczniki flagowe wstawiamy pomiędzy znakami procentu i konwersji. W przypadku liczby float: między % a f.
Przykładowe znaczniki:
.x (w miejscu ‘x’ jakaś cyfra) – ogranicza liczbę miejsc po przecinku,
- (pauza) – wyrównuje liczbę do lewej,
((lewy nawias) – ujmuje liczby ujemne w nawias,
, (przecinek) – grupuje cyfry,
+ (plus) – dodaje przed liczbami znak plusa,
(spacja) – dodaje odstęp do liczb nieujemnych

- Jak stworzyć archiwum zip ?
- Prosze wymienic i omowic typy prymitywne wystepujace w jezyku Java
- .Co zwraca metoda FileSystems.getDefault().newWatchService() ?

Klasy z biblioteki standardowej

- opisać typ newFile.Path, jak utworzyć obiekt takiego typu

- Path - z czym się to je?

Począwszy od Javy 1.7, bardziej zaawansowane operacje plikowe są dostępne poprzez klasy pakietu java.nio.file. Plik jest tutaj reprezentowany przez interfejs Path.
Path p1 = Paths.get("/tmp/file");
Path p2 = FileSystems.getDefault().getPath("/tmp/file");
Path p3 = Paths.get(URI.create("file:///tmp/file"));
lub
Path p = (new File("/tmp/file")).toPath();

- strumienie wejścia/wyjścia

strumienie - opisać wszystkie, jakie strumienie danych do poszczególnych strumieni, napisać na kartce program który czyta ze strumienia

Większość operacji wejścia/wyjścia wykorzystuje klasy pakietu java.io.
Strumienie bajtowe traktują dane jak zbiór ośmiobitowych bajtów. Wszystkie strumienie bajtowe rozszerzają klasy InputStream (dane przychodzące do programu) lub OutputStream (dane wychodzące z programu).
Strumienie zawsze należy zamykać!
Strumienie bajtowe reprezentują "niskopoziomowy" dostęp do danych. Dlatego w konkretnych sytuacjach warto je zastąpić przez bardziej specjalistyczne rodzaje strumieni.

```
FileInputStream in = null;
FileOutputStream out = null;
try {
        in = new FileInputStream("input.txt");
        out = new FileOutputStream("output.txt");
        int c;
        while ((c = in.read()) != -1) {
                out.write(c);
        }....
```

Strumienie znakowe automatycznie konwertują dane tekstowe do formatu Unicode (stosowanego natywnie w Javie). Konwersja jest dokonywana w oparciu o ustawienia regionalne komputera, na którym uruchomiono JVM (Wirtualną Maszynę Javy), lub jest sterowana "ręcznie" przez programistę. Strumienie znakowe rozszerzają klasy Reader (dane przychodzące do programu) lub Writer (dane wychodzące z programu).
Strumienie znakowe wykorzystuję do komunikacji strumienie bajtowe, a same zajmują się konwersją danych.
Strumienie znakowe buforowane umożliwiają odczytywanie tekstu linia po linii: Istnieją cztery klasy buforowanych strumieni: BufferedInputStream i BufferedOutputStream są strumieniami bajtowymi, podczas gdy BufferedReader i BufferedWriter odpowiadają za przesył znaków.
Aby wymusić zapis danych poprzez wyjściowy, buforowany strumień, można użyć metody flush().

```
FileReader in = null;
FileWriter out = null;
 try {
        in = new FileReader("input.txt");
        out = new FileWriter("output.txt");
        int c;
        while ((c = in.read()) != -1) {
                out.write(c);
        }.....
```

FileInputStream – przeznaczone do czytania SAMYCH BAJTÓW – np. obrazek
FileReader – do czytania ZNAKÓW

Scanner pozwala na przetwarzanie tokenów (domyślnie rozdzielonych przez Character.isWhitespace(char c)):
Obiekt należy zamknąć ze względu na strumień, z którym jest związany.

Strumienie binarne pozwalają efektywniej zarządzać zasobami. Istnieją dwa podstawowe rodzaje strumieni:
- strumienie danych: DataInputStream i DataOutputStream
- strumienie obiektowe: ObjectInputStream i ObjectOutputStream:

- java.nio.File

Klasa posiada również metody odpowiadające za podstawowe operacje na plikach: exists, move, delete, isReadable, … Można ją wykorzystać także do dostępu do plików.

- Serializacja

Serializacja to proces przekształcania obiektów, tj. instancji klas do postaci szeregowej czyli w strumień bajtów z zachowaniem aktualnego stanu obiektu. Serializowany obiekt może zostać utrwalony w pliku dyskowym, przesłany do innego procesu lub innego komputera przez sieć. Serializacja służy do zapisu stanu obiektu.

Podstawowym zastosowaniem strumieni obiektowych jest serializacja. Klasa wspierająca serializację musi implementować interfejs Serializable. Jeśli obiekty tej klasy wymagają specjalnego traktowania podczas serializacji należy zaimplementować metody:

private void writeObject(java.io.ObjectOutputStream out) throws IOException;
private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;
Dla obiektów typu JavaBeans istnieje także możliwość serializacji tekstowej (do plików w formacie XML) z wykorzystaniem klas XMLEncoder i XMLDecoder.

Tak jak data streams wspierają I/O typów prymitywnych, object streams wspierają I/O obiektów.
Klasy object stream:
ObjectInputStream
ObjectOutputStream
Te klasy implementują ObjectInput i ObjectOutput, które to są podinterfejsami DataInput i DataOutput. To oznacza, że object stream może zawierać zarówno prymitywne, jak i obiektowe typy.

ObjectInputStream deserializuje dane prymitywne i obiekty poprzednio napisane używając ObjectOutputStream.

inna odpowiedz

Serializacja jest podstawowym zastosowaniem strumieni obiektowych. Pozwala na przekształcenie obiektów w strumień bajtów (do postaci szeregowej) z zachowaniem aktualnego stanu obiektu. Serializowany obiekt może zostać utrwalony w pliku, przesłany do innego procesu lub innego komputera poprzez sieć. W skrócie serializacja służy do zapisu stanu obiektu a później do jego odtworzenia. By obiekt mógł być serializowany musi implementować interfejs Serializable. Dla wygody sporo standardowych klas Javy implementuje ten interfejs, nie ma więc potrzeby wyprowadzać np. własnego obiektu będącego dzieckiem klasy Vector. W przypadku serializacji obiektu, który agreguje inne obiekty, serializacji ulegnie cała hierarchia obiektów, jednak każdy z nich musi implementować interfejs Serializable. Interfejs ten nie wymaga implementacji żadnej metody. Każdy obiekt, który go zaimplementował może być serializowany/deserializowany do/ze strumienia. Jeśli obiekty danej klasy wymagają specjalnego traktowania podczas serializacji należy zaimplementować metody:
private void writeObject(java.io.ObjectOutputStream out) throws IOException;
private void readObject(java.io.ObjectInputStream in) throws IOException,
ClassNotFoundException.
Kod prezentujacy serializację na slajdach 18-20 z wykładu 5.

- Typy vectorow i roznice miedzy nimi (typy danych)

- kolekcja Properties + napisanie wczytywania z pliku do Properties

Properties to rozszerzenie tablicy haszującej.
public class Properties extends Hashtable<Object,Object>;
nastawione na przechowywanie par Stringów:
public synchronized Object setProperty(String key, String value);

- Co to są kolekcje (z przykładami):

Kolekcje są zbiorami danych. Najpopularniejszą kolekcją danych jest tablica. Jednak w wielu zastosowaniach przydatne są inne struktury danych jak listy, zbiory, mapy, tablice haszujące itp. Standardowa biblioteka Javy zawiera implementacje najpopularniejszych kolekcji w pakiecie java. Util. Ich podstawowa funkcjonalność jest zdefiniowana w interfejsie java.util.Collection.
Przykładowe kolekcje to:
Wektor (Vector) – w rzeczywistości to dynamiczna tablica, której rozmiar jest automatycznie dostosowywany do ilości danych. (więcej na slajdach 10-12 wykład 3)
Tablica haszująca (Hashtable) – kolekcja (mapa) zawierająca pary (klucz,wartość) Zarówno klucz jak i wartość mogą być dowolnymi obiektami. (Kod i przykład na slajdach 15-17 wykład 3)

- Jakie metody posiada interfejs Serializable?

- .Prosze omowic klase java.io.File

- Do czego służy klasa Scanner?

- Prosze omowic kolekcje java.util.Hashtable
- Prosze omowic kolekcje java.util.Properties
- Co to są kolekcje, wymienić i opisać przykładowe kolekcje

Typy generyczne

- deklaracja klasy generycznej implementująca interfejs generyczny
- Metody generyczne (Parametry ograniczone)
- printList(List<?> list) jakie argumentu może przyjmować
- Co oznacza deklaracja: <U extends Number> void inspect(U u);
- int extends Number? Number extends Number? Integer extends Number?
- Czy klasa java.util.Vector jest typem generycznym? Dlaczego? Czy przyjmuje typy proste (nie), powiedzieć coś o typach surowych.

- tworzenie tablicy typów generycznych

Jest niedozwolone dlatego, że tablica wymaga i identyfikuje typ swoich elementów podczas wykonania. Nakazując kompilatorowi stworzenie tablicy typu E kompilator będzie musiał wyprodukować kod tworzący tablicę, która nie zna typu swoich elementów, który gdyby go zapisać wyglądałby tak: new ?[]

- Metoda z typem generycznym <?> i dwia obiekty - jeden wbudowany i jeden generyczny, który pasuje do tej metody?

Znak '?' ogranicza 'od góry' rodzaj parametru.
przykład:
public static double sumOfList(List<? extends Number> list) { }
NIEPOPRAWNE:
public static double sumOfList(List<T extends Number> list)
Taka składnia jest dopuszczalna dla deklaracji klas lub wartości zwracanych (por. parametry ograniczone).

public static void printList(List<?> list) { - Znak '?' dopuszcza dowolny typ jako zawartość listy .
UWAGA: Metoda public static void printList(List<Object> list) nie obsłuży List<String> bo List<String> nie jest podklasą List<Object>.

- Czy są jakoś skorelowane vector<integer> i vector <number> / Klasa Integer jest potomkiem klasy Number. Jaka jest relacja pomiedzy klasami Vector<Integer> oraz Vector<Number>?

List<Integer> jest podtypem List<? extends Number>. Chodzi o dziedziczenie Number > Integer

- różnice generyczne i surowe, zalety wady

Można (ze względu na zachowanie zgodności z istniejącymi bibliotekami) stosować typy uogólnione jako typy surowe (ang. raw types), czyli pozbawione parametrów. Standard języka wyraźnie podkreśla, że ta możliwość jest jedynie tymczasowa.
Box<String> typedBox = new Box<String>();
Box rawBox = new Box(); // typ surowy <=> Box<Object>
Box rawBox = typedBox; // warning: przypisanie możliwe
rawBox.set(8); // warning: w trakcie działania programu może spowodować RuntimeException
Ze względu na możliwość popełnienia błędów, nie zaleca się stosowania typów surowych.

Wyjątki

Wyjątek w języku Java to obiekt, który opisuje pewną sytuację błędną lub nieprawidłową – wyjątkową. Jest to obiekt odpowiedniego typu, tj. obiekt klasy Throwable lub jej dowolnej podklasy.
Ważną częścią każdej aplikacji jest sygnalizowanie oraz obsługa pojawiających się błędów. Wystąpienie błędu w aplikacji zaimplementowanej w języku Java sygnalizujemy poprzez rzucenie wyjątku. Obsługa błędów to tzw. łapanie wyjątków.
Wyjątki dzielą się na dwa rodzaje: kontrolowane i niekontrolowane. Kontrolowane to takie które musimy deklarować i obsługiwać. Wyjątki niekontrolowane także możemy obsługiwać, ale nie musimy. Co to dokładnie oznacza dowiemy się niebawem.
Wyjątki niekontrolowane to instancje klas Error i RuntimeException oraz ich dowolnych podklas. Wyjątki kontrolowane to instancje klas Throwable i Exception oraz ich podklas, z wyłączeniem podklas klas Error i RuntimeException.
Wyjątki mogą być rzucane (zgłaszane) przez Wirtualną Maszynę Javy (JVM) oraz przez programistę.
Najpopularniejszym wyjątkiem rzucanym przez JVM jest NullPointerException. Wyjątek ten rzucany jest przy próbie odwołania się do obiektu (np. próba uruchomienia metody) z użyciem referencji, która ma wartość null.

NullPointerException jest podklasą klasyRuntimeException tak więc jest wyjątkiem niekontrolowanym – nie musimy go w żaden sposób deklarować czy obsługiwać.
Wyjątki możemy rzucać także sami. Robimy to, ilekroć chcemy zasygnalizować wystąpienie błędu. Możemy rzucać wyjątki standardowe, tj. instancje klas zdefiniowanych w bibliotece standardowej Java SE, ale możemy także definiować i rzucać wyjątki własne. Aby zdefiniować własny typ wyjątku wystarczy zaimplementować dowolną klasę będącą podklasą klasy Throwable lub dowolnej innej klasy definiującej wyjątek, np. klasy Exception czyRuntimeException. Wyjątki rzucamy używając słówka kluczowego throw.

INNA ODPOWIEDZ

Wyjątki są obiektami klasy Exception będącej klasą potomną Throwable. Wśród wyjatków znajduje się jedna szczególna klasa – RuntimeException, określająca błędy pjawiające się w trakcie działania programu, których nie można było łatwo przewidzieć na etapie tworzenia oprogramowania. np. NullPointerException, IndexOutOfBoundsException. Obsługa pozostałych wyjątków jest obowiązkowa tzn, czyli jeśli korzystamy z metody mogącej zwrócić wyjątek musimy wykonać jedną z dwóch czynności:
Obsłużyć wyjątek za pomocą bloku try...catch...(finally)
Zadeklarować, że nasza metoda może zwrócić ten wyjątek.
np. public void aMethod() throws FileNotFoundException{...}
Blok try...catch...(finally) próbuje początkowo wykonać instrukcje znajdujące się w części try, a jeśli pojawi się wyjątek zostaje on obsłużony za pomocą instrukcji w bloku catch. Opcjonalna część finally to kod który wykonuje się niezależnie od wystąpienia lub braku wystąpienia wyjątku.
Możliwe jest stworzenie własnego wyjątku poprzez rozszerzenie klasy Exception. Można przeciążyć metodę 'getMessage()' i za jej pomocą zwracać informacje o tym, co dokładnie się stało. Należy jednak napisać konstruktor dla wyjątku i wywołać w nim na początku konstruktor klasy Exception poprzez 'super()'
Taki wyjątek można rzucić przez 'throw new NazwaWyjątku()' należy jednak dopisać 'throws NazwaWyjątku' do metody mogącej go zwrócić. Później w kodzie należy go złapać albo przerzucić.

- Jak obsługujemy wyjątki w Javie

Wyjątki są rzucane i propagowane tak długo jak długo pozostają nie przechwycone. Wyjątki nie przechwycone propagują się aż do metody main(...) i - jeśli także tam nie są przechwycone - powodują przerwanie wykonania programu.
Aby przechwycić wyjątek rzucany przez pewien fragment kodu należy ten kod ując w klauzulę try-catch-finally. Składnia poniżej:

```
try {
  {kod programu}
} catch( {deklaracja zmiennej dla obiektu wyjątku} ) {
  {kod obsługi błędu}
} finally {
  {kod wykonywany zawsze}
}
```

Kod którego wyjątki chcemy przechwytywać to {kod programu}. Element {deklaracja zmiennej dla obiektu wyjątku} to deklaracja zmiennej typu takiego jak obsługiwany wyjątek do której będzie przypisany obiekt rzuconego i przechwyconego wyjątku. Wewnątrz klauzuli catch umieszczamy kod obsługi błędu - kod ten będzie wykonany tylko i wyłącznie wówczas gdy kod{kod programu} rzuci wyjątek typu zgodnego z typem określonym w deklaracji

{deklaracja zmiennej dla obiektu wyjątku}. Kod umieszczony wewnątrz klauzuli finally będzie wykonany zawsze, niezależnie od tego czy {kod programu} rzucił wyjątek czy nie.
Klauzula finally jest opcjonalna, za to klauzul catch może być dowolnie wiele. Może zdarzyć się - i często się zdarza - że kod ujęty w pojedynczej klauzuli try zwraca kilka różnych typów wyjątków. Możemy wtedy umieścić osobne klauzule catch dla obsługi każdego z tych wyjątków, ale możemy też umieścić jedną klauzulę catch do obsługi ich wszystkich - wystarczy, że korzystając z własności polimorfizmu (patrz artykuł "Polimorfizm") określimy typ wyjątku jakoException.

- Czy metoda main może deklarować throws Wyjątek

Odp. Wyżej.

- Blok finally

Odp. Wyżej.

- Czy można zrobić własny wyjątek i jeśli tak to jak?

Wątki

- Jak utworzyć proces w javie?

- powiedzieć coś o wait(), notify(), notifyAll()

Po pierwsze wolno tych metod używać tylko w sekcji synchronized, tudzież w metodach synchronizowanych i wołać je wolno tylko z obiektów na których odbywa się synchronizacja.

Można zatrzymać wątek i po pewnym czasie uruchomić go ponownie. Służy do tego konstrukcja wait - notify (wątek czeka na powiadomienie o jakimś zdarzeniu i powraca do stanu Runnable po powiadomieniu za pomocą notify) .

- Wątek wywołuje metodę wait.
- wait blokuje wątek (przesuwa go do stanu Not Runnable) i jednocześnie otwiera rygiel umożliwiając innym wątkom dostęp do obiektu. Inny wątek może zmienić stan obiektu i powiadomić o tym wątek czekający (za pomocą notify lub notifyAll).
- Wywołanie notify przywraca stan Runnable.

Metody sleep() oraz wait() wstrzymują wątek ale sleep() nie zwalnia blokady. wait() zwalnia blokadę i inne metody synchronizowane w obiekcie wątku mogą być wywoływane podczas oczekiwania.
UWAGA:
Metoda notify() wznawia jeden wątek wstrzymany na wait() jednak NIE oznacza to, że wątek ten zaraz zacznie się wykonywać i wejdzie ponownie do monitora - mogą go w tym uprzedzić inne wątki - nasz obudzony wątek współzawodniczy ze wszystkimi innymi wątkami próbującymi zająć monitor.
UWAGA:
NotifyAll() tym się różni od notify() że budzi wszystkie wątki wstrzymane na wait() - jednak monitor może zająć tylko jeden wątek!

- Co to jest blokada drobnoziarnista, po co się używa i jak?

Zamek (lock) jest przydatny wtedy, gdy operacje zamykania/otwierania nie moglyby byc umieszczone w jednej metodzie lub bloku synchronized.
Przykladem jest zakladanie blokady (lock) na elementy struktury danych, np. listy.
Dzieki temu unikamy koniecznosci blokowania calej listy i wiele watkow może rownocześnie przeglądac i modyfikowac rozne jej fragmenty.

Dostęp do c1 i c2 musi być synchronizowany niezależnie - nie chcemy blokować na raz obu liczników.

```
public class FineGrainedLockEx {
 private long c1 = 0, c2 = 0;
 private Object lock1 = new Object();
 private Object lock2 = new Object();
 public void inc1() {
            synchronized(lock1) {
                        c1++;
            }
}
 public void inc2() {
            synchronized(lock2) {
                        c2++;
            }
 }
}
```

- synchronize

odp wyzej.

- ProcessBuilder, omówić, powiedzieć po co się go stosuje

Proces można stworzyć korzystając z metody Runtime.exec() lub klasy ProcessBuilder z pakietu java.lang. Klasa ProcessBuilder pozwala precyzyjniej określić środowisko,
w którym działa proces. Można np. utworzyć funkcję sprawdzającą listę procesów.

- Co robi metoda join w wątkach?

Czeka na zakończenie wątku.
Wątek kończy swoje działenie w sposób "naturalny", gdy kończy się wykonywanie metody run(). Można też wątek przerwać "z zewnątrz", korzystając z jednej z metod:
- Thread.interrupt() - przerywanie wykonywania wątku
- Thread.join() - kończenie wątku

- Runnable

Można tworzyć wątki w oparciu o klasę implementującą interfejs java.lang.Runnable .
Jedyne co ten interfejs definiuje to metodę (void run()), która wywoływana jest przez wątki realizujące zadanie.

- ForkJoinPool jak działa, po co się stosuje i napisać jakiś przykładowy kod

Executor - jest to interfejs, którego implementacje odpowiadają za przyjęcie do realizacji zadania. Oddziela on mechanizm przyjmujący zadanie od mechanizmu wykonującego zlecone zadanie. Implementacje interfejsu muszą dostarczać jedną metodę:void execute(Runnable command).
Egzekutory oddzielają wątek od zarządzania nim

```
Executor exec = ...;
executor.execute(new RunnableTask1());
executor.execute(new RunnableTask2());
```

Executor jest rozszerzany przez ExecutorService.

Kolejnym rodzajem egzekutorów są tzw. Thread Pools – zbiory, wspólnie zarządzanych wątków. W JRE 7.0 wprowadzono ForkJoinPool,

ForkJoinPool - przeznaczony do wieloprocesorowych obliczeń równoległych.

- Blokada wewnętrzna

Każdy obiekt posiada powiązaną z nim blokadę wewnętrzna (intrinsic lock). Jeśli wątek chce uzyskać wyłączny dostęp do obiektu (lub jego atrybutów) może skorzystać z tej blokady.
Wbudowany mechanizm blokady – blok:
synchronized(referencja do obiektu, który służy jako blokada){
blok chroniony blokadą
}
Każdy obiekt posiada blokadę wewnętrzną zwaną także monitorującą (automatycznie zakładaną i zwalnianą) działającą jak muteks (jeden wątek ma „klucz do zamka")

Sieć

Standardowa obsługa sieci w Javie opiera się o tzw. interfejs gniazd.
Najważniejsze klasy, umożliwiające komunikację poprzez sieć to: Socket i ServerSocket

- co to jest socket?

Socket – klasa reprezentująca gniazdo służące do nawiązywania połączenia, wysyłania I odbierania danych, umożliwia połączenie pomiędzy klientem i serwerem
Klasa Socket pozwala na połączenie sieciowe na podanym porcie pomiędzy dwoma maszynami.
klasa Socket dla gniazda komunikacji strumieniowej peer-to-peer

Konstruktor tworzący gniazdo wygląda tak:
Socket socket = new Socket("komputer_w_sieci", 37);
Konstruktor tworzy gniazdo połączone z komputerem komputer_w_sieci na porcie 37 (komputer z którym chcemy się połączyć powinien słuchać na wybranym przez nas porcie).
Do odczytu i wysyłania danych służy mechanizm strumieni a nie obiekty typu DatagramSocket. Przykładowe zainicjowanie strumienia wejściowego i wyjściowego pokazane jest poniżej:
PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new
InputStreamReader(echoSocket.getInputStream()));
Takie połączenie klas odpowiadających za strumienie umożliwia odczyt i zapis w standardzie Unicode. Wysłanie znaku przez tak zainicjowany strumień wyjściowy połączony z gniazdem jest proste. Wystarczy wywołać metodę println(String text) w stosunku do strumienia wyjściowego:
out.println(text)
Odczytać dane z gniazda można na przykład wywołując metodę readLine() w stosunku do strumienia wejściowego.
Kończąc połączenie należy zamknąć zarówno gniazdo jak i strumienie wywołując metodę close() dla każdego z tych obiektów. Najpierw należy zamknąć strumienie a dopiero później gniazdo.

- wszystko o ServerSocket, jak działa, jak wygląda

ServerSocket – klasa reprezentująca gniazdo oczekujące na przychodzące żądania połączeń

klasa ServerSocket dla gniazda nasłuchu po stronie serwera
Do obsługi połączeń po stronie serwera JAVA udostępnia klasę ServerSocket, która posiada dodatkowe funkcjonalności umożliwiające równoczesną obsługę wielu połączeń. Tworzenie gniazda typu ServerSocket odbywa się tak samo jak gniazda Socket:
serverSocket = new ServerSocket(1234);
Jeśli wirtualna maszyna Javy nie będzie mogła zarezerwować wybranego portu dla tego gniazda to zostanie wyrzucony wyjątek IOException. Jeśli gniazdo uda się stworzyć, to kolejnym krokiem jest wywołanie metody accept(). Jest to metoda blokująca, która czeka na nadejście połączenia, a jeśli żądanie takie połączenia nadejdzie, to zwraca nowe gniazdo służące do obsługi tego połączenia. W tym samym czasie gniazdo typu ServerSocket może znów oczekiwać na nadejście połączenia.
clientSocket = serverSocket.accept();
Efektywne wspieranie wielu połączeń równocześnie wymaga na programiście zastosowania wątków. Dzięki temu może istnieć jeden proces, który oczekuje na przyjście połączenia i jeśli takie żądanie połączenia nastąpi to uruchamia oddzielny wątek dedykowany do obsługi tego żądania. Praktycznie wszystkie serwery sieciowe są wielowątkowe.

- dlaczego do tworzenia sslsocketow uzywamy fabryki a nie konstruktora?

Ponieważ wszystkie konstruktory tej klasy są chronione (??)

- SSLSocket - jak i po co się tworzy, napisać kod

Protokół SSL umożliwia bezpieczną (szyfrowaną) transmisję danych
poprzez niezabezpieczoną sieć.
Dodatkowo SSL umożliwia autoryzację
stron komunikacji. W tym celu wykorzystywany jest mechanizm
certyfkatów. Za transmisję z użyciem protokołu SSL odpowiedzialne są
klasy zgrupowane w pakiecie javax.net.SSL.
Domyślnie tylko jedna strona komunikacji (serwer) musi potwierdzać
swoją tożsamość. Aby wymusić autoryzację klienta należy użyć metod:
setNeedClientAuth(true) lub setWantClientAuth(true) wywołanych na
rzecz obiektu SSLServerSocket.

- Jak wczytac kod html strony internetowej? (napisać na kartce kod)

JNI

- jak napisać program w JNI?
- Jak wygenerować nagłówki JNI i jak wygląda taki przykładowy nagłówek.
- Czy funkcje natywne mogą być zmieniane w plikach innych języków programowania
- ODP: TAK, dowód Wykład 13, slajd: 18 + tłumaczenie tego kodu.
- omówić (*env)->ReleaseStringUTFChars(env, jin, in); tzn. jak działa i omówić co to są za argumenty
- konwersja z typów Javy na C

XML

- xml, roznice miedzy DOM a SAX, napisac cos w DOM

Dokument XML jest w całości wczytywany do pamięci. Dzięki temu DOM pozwala na manipulowanie danymi w całym dokumencie -ponieważ znajduje się on w pamięci.
However if the XML contains a large data, then it will be very expensive to load it into memory. Also the whole XML is loaded to memory although you are looking for something particular. You should consider using this technique, when you need to alter xml structure and you are sure that memory consumption is not going to be expensive.
Parsery DOM (Document Object Model) tworzą drzewo reprezentujące dane zawarte w dokumencie XML. To ułatwia pracę.

If you are creating a XML document (which is not big!) you should use this technique. However, if you are going to export a data from a database to xml (where you do not need navigation in the xml and/or data is huge) then you should consider other approaches.

SAX:
SAX (Simple API for XML) w miarę czytania dokumentu wywołuje zdarzenia związane z parsowaniem. Czyta dokument XML od początku do końca ale nie przechowuje niczego w pamięci.
Parsery SAX są szybsze i nie wymagają tak dużej ilości pamięci jak DOM.
Wadami interfejsów SAX, są niemożność uzyskania wszystkich danych z dokumentu XML (np. z komentarzy).
Poza tym w danej chwili parser nie ma dostępu do danych już przetworzonych.

- Napisac fragment kodu w SAX.
- DOM - napisać sposób użycia
- JAXB
- Jak stworzyc dokument w formacie XML, korzystajac z DOM?
- SAX
- Prosze napisac schemat kodu korzystajacego z SAX
- Do czego sluza klasy XMLDecoder i XMLEncoder?
- Jakich klas używa typowy program serwera?

Swing

Każda aplikacja wykorzystująca Swing używa co najmniej jeden kontener najwyższego poziomu.
- JFrame,
- JDialog,
- JApplet.

W takim kontenerze umieszcza się kolejne kontenery lub komponenty:

```
frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
// A tutaj tworzymy panel (kontener) dodajemy do niego komponenty
// a nastepnie umieszczamy go w kontenerze najwyzszego poziomu (frame)
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);
frame.getContentPane().add(contentPane)
```

Za rozmieszczenie komponentów w kontenerze odpowiedxzialny jest obiekt typu LayoutManager. Możemy go określić za pomocą metody setLayout(). Domyślnie kontenery używają instancji FlowLayout(), Jeśli chcemy "samodzielnie" określać pozycje komponentów należy usunąć LayoutManagera: container.setLayout(null)

Wszystkie komponenty rozszerzają klasę JComponent. Klasa JComponent implementuje następujące funkcjonalności:
- podpowiedzi (setToolTipText()),
- ramki (setBorder()),
- styl (UIManager.setLookAndFeel()),
- dodatkowe właściwości (setClientProperties()),
- rozmiary (layout)
- przystępność (accessibility)
- przeciągnij i upuść,
- podwójne buforowanie,
- wiązanie klawiszy.

- Co to jest TransferHandler i napisać z dokumentacją implementacje

Obiekt, który zarządza procesem drag'n drop jest TransferHandler. Posiada on metody służące do eksportu danych oraz importu.

```
// tworzy obiekt Transferable zawierający przenoszony element
 protected Transferable createTransferable(JComponent c) {
        Object obj = ((JList<File>) c).getSelectedValue();
        return new FileTranferable((File) obj);  }
```

- Opisać TransferHandler i TransferSupport.

importData(TransferHandler.TransferSupport) — metoda jest wywoływana po upuszczeniu (drop) obiektu. zwraca true jeśli import obiektu zakończył się powodzeniem.

- Transferable

Przenoszone dane są reprezentowane przez interfejs Transferable.

- jak ograniczyc file extensions w jfilechooser?
- JfileChooser - jak to wyglada, po co to jest i jak przechowywamy jest plik - czy w stringach czy w path

Pozwala na wyświetlanie okien dialogowych otwierania i zamykania plików.
Do przycisków „otwórz" i „zapisz" dodajemy actionListenery
mOtworz.addActionListener(this);
W ActionPerformed jeśli naciśnięto otwórz otwieramy okienko do wyboru pliku.
Później tworzymy plik File i tam wkładamy wybrany plik:
File plik = obiektJFileChooser.getSelectedFile();
Plik jest w File

- Prosze wymienic i krotko omowic komponenty tekstowe w bibliotece Swing?
- Prosze omowic klase JOptionPane.
- W jaki sposob filtrowac liste plikow wyswietlanych przez JFileChooser (np. wyswietlac tylko pliki *.txt)?
- Co sie dzieje z programem wyswietlajacym jedno okno JFrame, gdy uzytkownik zamknie to okno?

- Wycentrowanie elementów

frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);

Refleksje

Refleksja pozwala w łatwy sposób zarządzać kodem tak, jakby był danymi. Używa się jej najczęściej do zmieniania standardowego zachowania już zdefiniowanych metod lub funkcji, a także do tworzenia własnych konstrukcji semantycznych modyfikujących język. Z drugiej strony kod wykorzystujący refleksję jest mniej czytelny i nie pozwala na sprawdzenie poprawności składniowej i semantycznej w trakcie kompilacji (niewygodne śledzenie błędów).

Miłym zaskoczeniem jest to, że wszystkie klasy tworzone w JAVIE mogą być używane zarówno w sposób tradycyjny jak i refleksyjny. Dzięki temu nie musimy na etapie tworzenia deklarować przyszłego sposobu użycia klasy. Nie ma znaczenia ani modyfikator zakresu dostępu do pól czy metod jak i do samej klasy. Oczywiście klasa nie może być prywatna, gdyż wtedy nie będzie mogła być skompilowana jako główna klasa pliku JAVA. Ale poza tym nie ma żadnych ograniczeń - metody i pola mogą być publiczne, chronione czy też prywatne - będą mogły być użyte za pomocą refleksji w dokładnie taki sam sposób.

```
// bez refleksji
 Foo foo = new Foo();
 foo.hello();

 // z refleksją
 Class cl = Class.forName("Foo");
 Method method = cl.getMethod("hello");
 method.invoke(cl.newInstance());
```

Oba fragmenty tworzą instancję klasy Foo, następnie wywołują metodę hello() tej klasy. Różnica polega na tym, że w pierwszym fragmencie nazwa klasy i metody są częścią kodu źródłowego, podczas gdy w drugim fragmencie możliwe jest przeniesienie ich do zmiennych, których wartość jest ustalana w czasie wykonania kodu.

- mając do dyspozycji referencję Method m, do metody obj.metoda(obj2) wywołać metodę "metoda" poprzez wspomniana referencję m.

m.invoke(obj, obj2) = tylko od java 1.7 (jak dowiedziałem się od Cieśli), wcześniej lista argumentów musiała być w formie tablicy: m.invoke(obj, [obj2]).

- Dynamiczne proxy.
- Co to jest java.lang.Reflection.InvocationHandler?

- Refleksyjne ustawienie pola obiektu mając do niego referencję
- Jak wywołać metodę w programowaniu refleksyjnym. Jakie argumenty trzeba podać?
- Jak otrzymać instancję z Class?
- Co trzeba zrobic, zeby udostepnic program poprzez narzedzie JavaWebStart?
- Co robi metoda Class.forName() + jaki jest "efekt uboczny" jej wywołania?
- Jak, korzystajac z refleksji, wywolac metody (okreslona przez instancje klasy Method) przyjmujaca jako argument typ java.lang.Object?
- W jaki sposob utworzyc nowy obiekt bez uzycia operatora new?
- Czym jest java.lang.reflect.InvocationHandler i do czego sluzy?
- Co robi funkcja Class.forName(coś)
- Jak, korzystajac z refleksji, uzyskac informacje o publicznych konstruktorach w danej klasie? Jak pobrać jakiś konkretny konstruktor?
- Jak, korzystajac z refleksji, zmienic wartosc publicznego atrybutu o nazwie "i" w obiekcie obj?
- Omówić dynamiczne obiekty proxy (refleksja).

Programowanie niskopoziomowe, kompilacja, uruchamianie

- bytecode, specyfikacja pliku class
- co zawiera plik class i czy da sie go zdekompilować
- Skąd maszyna wirtualna wie, że w tej linii wystąpił błąd? Jak to jest zapisane w bytecode?
- java web start
- ANT

inne

- Co to jest .jar i jak się kompiluje. Co to jest META-INF oraz co zawiera plik MANIFEST.MF:

Java wyróżnia także szczególny rodzaj archwium ZIP: JAR (JarOutputStream, JarInputStream).
Archiwa JAR zawierają pliki klas wraz z dodatkowymi zasobami potrzebnymi do działania aplikacji. Podstawowe zalety dystrybucji programów w postaci plików .jar to:
bezpieczeństwo – archiwa mogą być cyfrowo podpisywane
kompresja – skrócenie czasu ładowania apletu lub aplikacji
zarządzanie zawartością archiwów z poziomu języka Java
zarządzanie wersjami na poziomie pakietów oraz archiwów (Package Sealing, Package Versioning)
przenośność
Archwium jar tworzy się używając komendy jar, np.:
jar cf archiwum.jar klasa1.class klasa2.class ….
c – tworzenie pliku
f – zawartość archwium zostanie zapisana do pliku archiwum.jar zamiast do standardowego wyjścia (stdout)
m – do archiwum zostanie dołączony plik manifest z określonej lokalizacji, np. jar cmf plik_manifest archiwum.jar*

W archiwum jar znajduje się katalog META-INF a w nim plik MANIFEST.MF zawierający dodatkowe informacje o archiwum. Przykładowa zawartość:
Manifest-Version: 1.0
Created-By: 1.5.0-b64 (Sun Microsystems Inc.)
Ant-Version: Apache Ant 1.6.5
Main-Class: pl.edu.uj.if.wyklady.java.Wyklad06
Mówi, że po uruchomieniu archiwum wykonana zostanie metoda main (String[] args) zawarta w klasie Wyklad06 znajdującej się w pamiekie pl.edu.uj.if.wyklady.java.
Uruchomienie pliku jar:
java -jar archiwum.jar

- Co to jest Java Architecture for XML Binding?

JAXB to standard serializacji XML dla obiektów Javy. Został on zintegrowany z JDK/JRE od wersji 1.6. Kod prezentujący działanie – wykład 11, slajdy od 12 do 14.

- Jak przekazuje się nieokreśloną ilość parametrów do metody?

Za pomocą '…' . Np. void foo(String... args)

- Blok inicjujący

wygląda jak metoda bez typu i nazwy. Może posłużyć do zainicjowania pól klasy:
Blok inicjalizacji statycznej:

```
static {
//jakiś kod
}
```

Blok inicjalizacji instancyjnej:

```
{
//jakiś kod
}
```

Blok inicjalizacji statycznej jest wykonywany tylko raz przy ładowaniu klasy.
Blok inicjalizacji instancyjnej jest wykonywany dla każdej instancji dokładnie raz (tak, jak konstruktor).
Można powiedzieć, że blok inicjujący zachowuje się jak konstruktor bez parametrów. W bloku inicjującym można zawrzeć operacje wykonywane zawsze.