Politechnika Gdańska, ***Inżynieria Biomedyczna***

Język JAVA
podstawy programowania

Jacek Rumiński

wykład 2, część 3

Plan wykładu:

1. Rodzaje programów w Javie
2. Tworzenie aplikacji
3. Tworzenie apletów
4. Obsługa archiwów
5. Wyjątki
6. Klasa w klasie !

Co to jest archiwum Javy (JAR – Java ARchive)?

Odpowiedź właściwie jest prosta: spakowane metodą ZIP pliki, wchodzące w skład biblioteki, aplikacji, apletu, ...
W JDK dostępne jest specjalne narzędzie jar.exe.

W celu stworzenia archiwum z wszystkich klas zawartych w bieżącym katalogu i nadać mu nazwę JediArchiwum można wywołać polecenie:

```
jar  -cf JediArchiwum.jar  *.class
```

-c **stwórz nowe archiwum**
-t **pokaż zawartość archiwum**
-x **pobierz podane pliki z archiwum**
-u **aktualizuj archiwum**
-f **określenie nazwy archiwum**

Archiwum – demonstracja.

Stwórzmy archiwum złożone z kilku plików *.class dla przykładowego apletu. Pliki źródłowe:

RycerzJedi.java
Sith.java
Rywalizacja.java (aplet)

Wszystkie przykładowe pliki programów załączono na stronie kursu (SZKOLENIA/JAVA) http://uno.biomed.gda.pl

Archiwa Javy

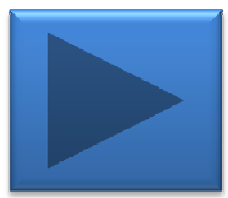
Warto zauważyć, że wszystkie dystrybuowane biblioteki Javy (zarówno w JDK jak i inne) dostarczane są najczęściej w formie plików JAR.

Jeśli chcesz zobaczyć co jest zawartością pliku archiwum można tymczasowo zmienić jego rozszerzenie na „.zip” i rozpakować (podgląd). Wówczas zobaczymy strukturę pakietów (czyli katalogów) oraz zawarte pliki klas (.class) i ewentualnie inne pliki (zasoby).

Archiwa Javy

Archiwa mają również ciekawe zastosowania dla potrzeb apletów. Otóż jeśli aplet składa się z wielu plików to lepiej utworzyć archiwum (które będzie przesłane jako jeden, skompresowany plik) i wywoływać główną klasę apletu względem archiwum.

```
<applet code="Rywalizacja.class"  archive="JediArchiwum.jar"
width="400" height="300"></applet>
```



Ponadto archiwum można podpisać cyfrowo i po weryfikacji nadać

Plan wykładu:

1. Rodzaje programów w Javie
2. Tworzenie aplikacji
3. Tworzenie apletów
4. Obsługa archiwów
5. Wyjątki
6. Klasa w klasie !

Wyjątki.

Wyobraźmy sobie następujące sytuacje:

- chcemy czytać z pliku o podanej nazwie, ale plik nie istnieje,
- wywołujemy połączenie z komputerem, ale adres jest nieprawidłowy,
- przesyłamy dane, ale zerwane zostało połączenie
- itp.

Sytuacje takie nie są prawidłowe czy pożądane. Jeśli możemy przewidzieć ich ***potencjalne*** wystąpienie to lepiej zabezpieczyć kod źródłowy (zamiast otrzymać w czasie wykonywania programu coś typu *General Protection Fault*).

Tym zabezpieczeniem jest oznaczenie określonych metod, że mogą potencjalnie być źródłem wyjątku.

Co to jest wyjątek?

Wyjątek to instancja uniwersalnego typu danych, który ma wartości i funkcje - czyli → wyjątek to obiekt, typ wyjątku to klasa!
Każdy wyjątek ma swoją nazwę (nazwę klasy), która zwykle sugeruje jakiego rodzaju jest to wyjątek, np.:

w pakiecie java.io.*:
FileNotFoundException - brak pliku
InterruptedIOException - przerwanie operacji we/wy
IOException - klasa nadrzędna wyjątków we/wy

w pakiecie java.lang.*:
ArrayIndexOutOfBoundsException - przekroczenie zakresu tablicy,
ClassNotFoundException - brak klasy,
Exception - klasa nadrzędna wyjątków.

Skąd mam wiedzieć kiedy i jak obsługiwać wyjątki?

Jeśli używasz jakiejś metody, która może zwrócić wyjątek wówczas musisz obsłużyć wyjątek lub odpowiednio wprowadzić właściwą deklarację.

Jeśli tego nie zrobisz (a wyjątek jest innego rodzaju niż RuntimeException czy Error) wówczas kompilator poinformuje Cię, że kompilacja się nie udała, bo brak wymaganej obsługi (deklaracji) wyjątku. Ponadto poda informacje o jaki wyjątek (nazwa klasy) chodzi i dla jakiej metody powinna być napisana obsługa wyjątku.
Zawsze możemy sprawdzić w dokumentacji klas, czy dana metoda zwraca wyjątek.

Możemy tworzyć również własne typy wyjątków (czyli klasy) i generować je (stwarzać i przesyłać obiekty).

Skąd mam wiedzieć kiedy i jak obsługiwać wyjątki?

Obsługa wyjątku:

```
try{
      //1.tu wywołanie metody, która może wywołać wyjątek klasy Exception
      //2. jeśli dojdziemy do drugiej linii kodu po 1. – wyjątek nie pojawił się
} catch (Exception e) {
     //tu instrukcje związane z obsługą sytuacji wyjątkowej (np. komunikat)
```

Deklaracja (określanie) wyjątku:

```
public void mojaMetoda() throws Exception{
       //(...)
      //1.tu wywołanie metody, która może wywołać wyjątek klasy Exception
      //(...)
}
```

Jak obsługiwać wyjątki?

Jedna metoda zwraca więcej niż jeden wyjątek?

```
try{
      //1.tu wywołanie metody, która może wywołać wyjątki klas:
      // Exception1  i Exception2
      //2. jeśli dojdziemy do drugiej linii kodu po 1. – wyjątek nie pojawił się
} catch (Exception1 e1) {
     //tu instrukcje związane z obsługą wyjątku Exception1
}catch (Exception2  e2){
    //tu instrukcje związane z obsługą wyjątku Exception2
} finally{ //opcjonalne
   //tu instrukcje, których wykonanie jest gwarantowane , np. zamknięcie
   //połączenia sieciowego
}
```

Kiedy nie chcemy obsługiwać wyjątku -

ponieważ ma być obsługiwany przez metody, korzystające z danej metody zwracającej wyjątek - wówczas deklarujemy możliwość pojawienia się wyjątku poprzez użycie słowa kluczowego „throws” i wyliczenie (po przecinkach) klas wyjątków:

```
public void mojaMetoda() throws Exception{
        //(...)
       //1.tu wywołanie metody, która może wywołać wyjątek klasy Exception
       //(...)
```

Jeśli ktoś będzie chciał użyć metody mojaMetoda() wówczas musi albo obsłużyć wyjątek albo zadeklarować throws ...itd.

Jak samemu generować wyjątki ?

Do tego służy specjalne słowo kluczowe (instrukcja) „throw”.

```
//(...)
if (liczbaOdebranychPakietow < 10) //jeżeli  nie spełniony jest warunek
        throw new Exception("Za malo odebranych pakietow!!!");
//(...)
```

Klasa Exception to jedna ze standardowych klas wyjątków (jedna z najbardziej ogólnych, po której większość dziedziczy).
Jeśli chcemy mieć własny typ wyjątku, np. MyException wówczas należy stworzyć nową klasę dziedzicząc po klasie np. Exception:

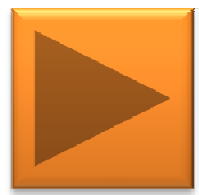
```
public class MyException extends Exception{
        //tu treść – jak dla każdej klasy – o czym dalej
} //koniec class MyException
```

Wyjątki – demonstracja.

WyjatkowyJedi.java
MyException.java

DEMO

Wszystkie przykładowe pliki programów załączono na stronie kursu (SZKOLENIA/JAVA) http://uno.biomed.gda.pl



Plan wykładu:

1. Rodzaje programów w Javie
2. Tworzenie aplikacji
3. Tworzenie apletów
4. Obsługa archiwów
5. Wyjątki
6. Klasa w klasie !

Co jeszcze może zawierać klasa?

Oprócz cech (pól), konstruktora oraz metod klasa może zawierać jeszcze klasy wewnętrzne.
Klasę wewnętrzną definiujemy w obrębie nawiasów klamrowych klasy zewnętrznej.

```
public class MasterJedi{
        //pola i funkcje klasy MasterJedi
        class StudentJedi {
                //pola i funkcje klasy StudentJedi
        }//koniec class StudentJedi
}//koniec class MasterJedi
```

Klasa wewnętrzna staje się częścią klasy zewnętrznej i dostęp do niej jest

Język JAVA –klasy wewnętrzne


```
public class MasterJedi {
        public long mPesel;
        public MasterJedi(long pes) {
                mPesel = pes;
        }
        class StudentJedi {
                public long sPesel;
                public StudentJedi(long pes)  {
                        sPesel=pes;
                        System.out.println("PESEL mistrza to: " +mPesel);
                        System.out.println("PESEL ucznia to: " +sPesel);
                };
        }//koniec class StudentJedi
         public static void main(String args[]) {
           MasterJedi mj = new MasterJedi(90100903890L);
           MasterJedi.StudentJedi sj= mj.new StudentJedi(80081204591L);
         }//koniec main()
}//koniec public class MasterJedi
```

Klasy wewnętrzne

Klas wewnętrznych używamy – rzadko – najczęściej wtedy, gdy chcemy zgrupować klasy, które są stosowane tylko w jednym, danym miejscu. Klasy wewnętrzne mogą być dynamiczne (jak w poprzednim przykładzie) lub statyczne (czyli należą do klasy) oznaczone specyfikatorem static .

Klasy wewnętrzne dynamiczne mogą bezpośrednio odwoływać się do wszystkich pól i metod klasy zewnętrznej!

Klasy wewnętrzne statyczne mogą to zrobić wyłącznie przez

```
//w przykładzie MasterJedi.java:
public StudentJedi(long pes)  {
  sPesel=pes;
  System.out.println("PESEL mistrza to: " +mPesel); //odwołanie do pola klasy zew.
  System.out.println("PESEL ucznia to: " +sPesel);
};
```

Klasy wewnętrzne

Do klas wewnętrznych powrócimy omawiając obsługę zdarzeń związanych z graficznym interfejsem użytkownika (czyli np. obsługę wciśnięcia przycisku).

A teraz – zapraszam na kolejny wykład nr 3, w którym zapoznamy się z zasadami budowy zmiennych, poznamy typy danych oraz podstawowe instrukcje sterowania programem jak pętle, instrukcje warunkowe, itp.

Zapraszam na wykład 3