Informacje ogólne (ostatnia aktualizacja 9 X 2011)

Zasady zaliczenia

Każde z zadań ma oznaczony termin oddania. Student może uzyskać maksymalną liczbę punktów za dostarczenie rozwiązania zadania, o ile zostanie ono zaprezentowane najpóźniej w trakcie zajęć w określonym terminie oddania. Prezentacja rozwiązania najpóźniej w trakcie zajęć następujących po terminie daje możliwość uzyskania maksymalnie 75% punktów. Ostatnim terminem dającym możliwość uzyskania co najwyżej 50% punktów za prezentowane rozwiązanie są kolejne zajęcia – tj. drugie w kolejności po terminie pierwotnym.

W trakcie prezentacji student musi wykazać się znajomością oddawanego rozwiązania świadczącą o jego samodzielnym wykonaniu. **W przypadku, gdy prowadzący na podstawie pytań kontrolnych dojdzie do przekonania, że prezentowane rozwiązanie nie zostało wykonane samodzielnie student nie uzyskuje żadnych punktów oraz bezpowrotnie traci szansę ponownej próby oddania rozwiązania.**

Testy jednostkowe

Stworzonej funkcjonalności powinny towarzyszyć testy jednostkowe weryfikujące poprawność jej działania. Brak testów jednostkowych ogranicza zmniejsza liczbę uzyskanych punktów o połowę.

Odporność na błędy

Dostarczone aplikacje powinny w miarę możliwości przeprowadzać walidację danych wejściowych, tj. weryfikować poprawność danych wprowadzanych przez użytkownika. Do niedopuszczalnych zachowań skutkujących drastyczną redukcją liczby uzyskanych punktów należą m.in.:

- brak weryfikacji, czy użytkownik podał poprawną ścieżkę do pliku i czy plik/katalog istnieje;
- brak walidacji formatu wprowadzonych danych – jeśli format jest znany z góry (np. format zapisu kodu pocztowego);
- brak weryfikacji, czy wartość referencyjnej zmiennej/pola została ustawiona, tj. czy nie ma wartości null (NullPointerException);
- brak weryfikacji, czy wartość indeksu nie przekracza rozmiaru listy/tablicy (ArrayIndexOutOfBoundsException).

Zadania obowiązkowe

Za realizację minimum programowego z przedmiotu UTP uprawniającego do otrzymania oceny pozytywnej uznaje się zaliczenie (tj. uzyskanie co najmniej 50% punktów) zadań poświęconych następującym zagadnieniom:

- strumienie (zadanie II);
- kolekcje (zadanie IV);
- JavaBeans (zadanie VII);
- JDBC (zadanie XI).

1 Zadanie I – Model-View-Controller (10 X 2011/13 X 2011)

Stwórz prostą aplikację, której jedynym elementem interfejsu użytkownika będzie tabela zawierająca jedną kolumnę umożliwiającą prezentację oraz edycję kodów pocztowych w formacie obowiązującym w Polsce, tj. 00-000.

Edycja wartości powinna być oparta o pole maskowane zrealizowane w oparciu o JFormattedTextField oraz MaskFormatter.

Aplikacja powinna być stworzona zgodnie ze wzorcem Model-View-Controller – tabela prezentująca kody pocztowe powinna być oparta o model zawierający listę instancji typu PostalCode hermetyzującego kody pocztowe.

2 Zadanie II – strumienie (17 X 2011/20 X 2011)

Stwórz prostą aplikację, której graficzny interfejs użytkownika składa się z dwóch wieloliniowych pól tekstowych (JTextArea):

- pole tekstowe (INPUT) znajdujące się po lewej stronie umożliwia użytkownikowi wprowadzanie danych;
- z kolei drugie pole tekstowe (OUTPUT) w trybie tylko do odczytu prezentuje wynik przetwarzania wejścia wprowadzonego za pośrednictwem pola tekstowego INPUT.

Pole tekstowe INPUT umożliwia wprowadzanie tekstu – linia po linii, lub nazwy pliku, która powinna być podana w formacie bezwzględnej ścieżki używanej w systemie operacyjnym Windows lub Un*x.

Pole tekstowe OUTPUT prezentuje wynik przetwarzania linii wprowadzonej za pomocą pola INPUT. Przetwarzanie polega na zliczeniu występowania danego (zaszytego w kodzie aplikacji) słowa, bądź wyrażenia regularnego w linii wprowadzonej przez użytkownika, bądź też pliku – jeśli użytkownik wprowadził poprawną ścieżkę do pliku. Poprawność ścieżki do pliku można wstępnie sprawdzić za pomocą wyrażenia regularnego oraz metody exists() klasy File.

Rozwiązanie można rozszerzyć o dodanie kolejnego pola tekstowego (JTextField), które umożliwi użytkownikowi wprowadzenie szukanego słowa/wyrażenia regularnego.

Największą trudnością zadania jest konieczność odczytu zawartości pliku do bufora, w którym następnie należy wykonać wyszukiwanie określonego ciągu znaków, bądź dopasowanie do wskazanego wyrażenia regularnego.

Student może uzyskać łącznie 15 punktów za rozwiązanie, w którym zostanie zastosowana znana z parserów technika polegająca na (1) przepisywaniu drugiej połowy bufora do pierwszej, (2) wczytywaniu kolejnego fragmentu pliku do drugiej połowy, a następnie (3) sprawdzaniu czy cały bufor zawiera szukany ciąg.

Proste rozwiązania polegające na zaczytaniu całości wejścia do pamięci będą oceniane na co najwyżej 10 punktów.

3 Zadanie III – kodowanie/dekodowanie (24 X 2011/27 X 2011)

Stwórz prostą aplikację graficzną umożliwiającą konwersję kodowania plików tekstowych.

Interfejs użytkownika będzie zawierał dwa pola tekstowe (JTextField) umożliwiających określenie ścieżek do plików: (1) wejściowego i (2) wyjściowego. Alternatywnie można określić ścieżkę do pliku przy pomocy okna dialogowego (JFileChooser) – wówczas pola tekstowe mogą być ustawione w trybie tylko do odczytu, bądź mogą być zastąpione etykietami.

Poniżej każdego z pól tekstowych powinna znajdować się lista opuszczana (JComboBox) zawierająca listę wspieranych stron kodowych (patrz) umożliwiająca określenie strony kodowej odpowiednio dla pliku wejściowego i wyjściowego.

4 Zadanie IV – kolekcje (7 XI 2011/10 XI 2011)

Stwórz aplikację, w której wykorzystasz różne rodzaje kolekcji dostępne w ramach Java™ Collections Framework. Aplikacja powinna odczytywać dane wejściowe z pliku. Format danych zapisanych wejściowych powinien być następujący:

```
<Imię> <Nazwisko> <Data urodzenia>
<Imię> <Nazwisko> <Data urodzenia>
<Imię> <Nazwisko> <Data urodzenia>
```

Przykładowo:

```
Jan Kowalski 1980-01-01
Henryk Nowak 1982-03-03
```

Oczywiście każdej osoby powinny być reprezentowane w aplikacji w postaci instancji własnej klasy Person.

Dane wejściowe powinny być zapisane w kilku kolekcjach:

1. Posortowanej po imieniu;
2. Posortowanej po nazwisku, imieniu i dacie urodzenia;
3. Posortowanej po dacie urodzenia;
4. Kolekcji umożliwiającej szybki dostęp do rekordu po dacie urodzenia wprowadzonej w postaci typu java.util.Date, bądź w postaci java.lang.String w formacie YYYY-MM-DD.

Aplikacja powinna posiadać interfejs graficzny składający się z:

- pola tekstowego (JTextField) umożliwiającego wprowadzenie ścieżki do pliku;
- listy (JList) lub tabeli (JTable) prezentującej wybraną kolekcję – przełączanie między kolekcjami może być zrealizowane w dowolny sposób – np. za pomocą menu wyskakującego (JPopupMenu), przycisków (JButton), listy opuszczanej (JComboBox).

5 Zadanie V – kolekcje c.d. (14 XI 2011/17 XI 2011)

Stwórz aplikację wielowątkową opartą o architekturę zlecający-usługa (*requestor-service*). Wątki klienckie (*requestor threads*) umieszczają w kolejce zadania realizowane przez wątki usług (*service threads*). Usługi pobierają zlecenie z kolejki, realizują je, a następnie umieszczają w kolejce z tym samym priorytetem, z którym zostało zapisane w kolejce przez zleceniodawcę. Ponieważ użytkownik powinien mieć możliwość przypisania zlecanym zadaniom priorytetów, rozwiązanie powinno być oparte o klasę PriorityQueue.

Przykładowym zadaniem realizowanym przez wątki usługowe może być dodawanie dwóch wartości. Zlecenie takiego zadania powinno zawierać dwa składniki dodawania, które powinny zostać uzupełnione o sumę dodawania w zadaniu zrealizowanym.

Aplikacja powinna posiadać interfejs graficzny składający się z listy (JList) reprezentującej kolejkę zadań – lista powinna być odświeżana po każdym: (1) dodaniu zadania do kolejki, (2) usunięciu zadania z kolejki. Każde zadanie powinno mieć unikalny identyfikator oraz status umożliwiający stwierdzenie, czy jest to zadanie zlecone, czy zrealizowane przez usługę.

6 Zadanie VI – refleksja (21 XI 2011/24 XI 2011)

Jedną z dobrych praktyk podczas tworzenia oprogramowania jest hermetyzacja (*encapsulation*) mająca na celu ukrycie złożoność implementacji przed użytkownikiem – programistą korzystającym z API. Wadą hermetyzacji jest brak możliwości weryfikacji poprawności działania ukrytej funkcjonalności za pomocą testów jednostkowych.

Sposobem na uzyskanie do prywatnych (*private*), chronionych (*protected*), pakietowych-prywatnych (*package-private*) składowych klas jest użycie mechanizmu refleksji.

Stwórz klasę reprezentującą numer PESEL (Powszechny Elektroniczny System Ewidencji Ludności) posiadającą:

- prywatną metodę walidacyjną zgodnie z zasadami opisanymi w http://pl.wikipedia.org/wiki/PESEL;
- prywatną metodę ustalającą wartość daty urodzenia na podstawie pierwszych 6-iu cyfr numeru PESEL;
- prywatną metodę ustalającą płeć – wartość wyliczalną (*enum*) – na podstawie 10-ej cyfry numeru PESEL.

Zweryfikuj poprawność działania wymienionych metod za pomocą testów jednostkowych wywołujących metodę za pomocą refleksji.

7 Zadanie VII – JavaBeans (28 XI 2011/1 XII 2011)

Stwórz prostą aplikację, której interfejs graficzny użytkownika składa się z 3 pól tekstowych umieszczonych jedno pod drugim. Wszystkie z wymienionych pól tekstowych aplikacji powinny umożliwiać użytkownikowi wprowadzenie wyłącznie liczb całkowitych.

Pole tekstowe nr 1 powinno dawać możliwość określenia maksymalnej wartości różnicy odejmowania zawartości pola nr 3 od pola nr 2. W przypadku, gdy po zmianie wartości pól 2 lub 3 różnica ta jest zbyt duża ostatnio wprowadzona przez użytkownika zmiana jest automatycznie wycofywana oraz prezentowany komunikat, że różnica między wartościami pól 2 i 3 przekracza dozwolony limit. Brak określenia wartości pola tekstowego 1 powinno wiązać się z wyczyszczeniem wartości pól 2 i 3 oraz zablokowaniem możliwości ich edycji.

Twoja aplikacja powinna być stworzona zgodnie z zasadami wzorca projektowego Model-View-Controller i w oparciu o mechanizm JavaBeans. Model aplikacji powinien składać się z 3 właściwości liczbowych. Pola tekstowe powinny być widokiem prezentującymi bieżące wartości wspomnianych właściwości. Zmiana wartości któregokolwiek z pól powinna wyzwalać kontroler próbujący odpowiednio zmienić wartości właściwości. Wymuszenie ograniczenia na różnicę wartości pól 2 i 3 powinno być realizowane za pomocą mechanizmu wetowania JavaBeans.

8 Zadanie VIII – adnotacje (5 XII 2011/8 XII 2011)

Stwórz prostą aplikację demonstrującą wykorzystanie adnotacji dostępnych w Java™ w czasie wykonania.

Interfejsem graficznym aplikacji powinna być tabela (JTable) bazująca na modelu zgodnie z paradygmatem MVC. Model powinien określać kolejność kolumn na podstawie wartości właściwości index(), którym powinna być opatrzona każda metoda dostępowa *get* klasy, których instancje są prezentowane w tabeli.

W celu zapewnienia wydajności wykreślania (*render*) tabeli należy zadbać, by dla każdego typu elementów tabeli kolejność kolumn była ustalona jednorazowo – w trakcie tworzenia pierwszego modelu w oparciu o dany typ. Niedopuszczalne jest określanie indeksu kolumny na podstawie refleksji każdorazowo w trakcie wykreślania komórek tabeli.

```
@Retention(RetentionPolicy.RUNTIME)
public @interface ColumnMetadata {
   int index();

public class Person {
   ...
   @ColumnMetadata(index=0)
   String getFirstName() { ... }

   @ColumnMetadata(index=1)
   String getLastName() { ... }

   @ColumnMetadata(index=2)
   String getBirthDate() { ... }
   ...
}
```

W trakcie tworzenia modelu należy oczywiście zweryfikować, czy wartości właściwości index() (1) są unikalne.

Trudniejszą wersją zadania jest weryfikacja, o której mowa powyżej na etapie kompilacji za pomocą Annotation Processing Tool. Realizacja zadania w wersji trudniejszej wraz z zapewnieniem pełnej automatyzacji procesu budowy za pomocą Apache Ant™ umożliwia uzyskanie dodatkowych 10 pkt.

Kolejnym utrudnieniem, i jednocześnie usprawnieniem rozwiązania byłoby generowanie klasy modelu tabeli na podstawie szablonu w trakcie kompilacji. Realizacja najtrudniejszego wariantu zadania wraz z automatyzacją kompilacji za pomocą Apache Ant™ uprawnia do uzyskania łącznie 30 pkt.