Wrocław, 11 marca 2024

Platformy **P**rogramistyczne `.NET` i `Java`

Laboratorium 5

Projekt aplikacji okienkowej `Java` na przykładzie prostego problemu optymalizacyjnego

prowadzący: *Dr inż. Radosław Idzikowski, mgr inż. Michał Jaroszczuk*

1 Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawami projektowania aplikacji konsolowej i okienkowej w języku programowania `Java` na przykładzie nieograniczonego problemu plecakowego. Obejmuje to odpowiednie zdefiniowanie problemu, danych wejściowych oraz implementację prostego algorytmu, a także napisanie przejrzystego graficznego interfejsu użytkownika (*ang. graphical user interface*, GUI). Wybór IDE jest dowolny, natomiast zaleca się użycie programu `IntelliJ` (przykłady w niniejszej instrukcji będą tworzone z użyciem tego środowiska). **Ukończenie każdego etapu powinno być zgłoszone prowadzącemu w celu akceptacji i odnotowania postępów**. Sam program należy umieścić na repozytorium `github` i wysłać zaproszenie do prowadzącego. Czas na wykonanie zadania to dwa zajęcia laboratoryjne. Podczas pierwszego spotkania trzeba wykonać co najmniej jedno zadanie zdefiniowane w Rozdziale 2.

2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Pierwsza aplikacja konsolowa `Java` – optymalizacja nieograniczonego problemu plecakowego.
2. Testy jednostkowe dla aplikacji konsolowej.
3. Graficzny interfejs użytkownika.

Za wykonanie zadania nr 1 jest ocena dostateczna, za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. Link do repozytorium należy przesłać dopiero po oddaniu i ocenieniu pracy na laboratorium.

3 Opis zadań

3.1 Zadanie 1

W ramach tego zadania będzie do wykonana nasza pierwsza aplikacja w technologii `Java`. Warto samemu poszerzyć wiedzę na temat założeń i koncepcji dotyczących tego języka, gdyż ich znajomość jest niezbędna do poprawnego wykorzystania jego możliwości. Na początek należy uruchomić program `IntelliJ` oraz utworzyć nowy projekt wybierając kolejno:

1. Nazwę projektu,

2. Lokalizację projektu,

3. Język, w którym pisany będzie projekt (dostępne są również inne języki wywodzące się z Javy). W przypadku niniejszego projektu należy wybrać Javę.

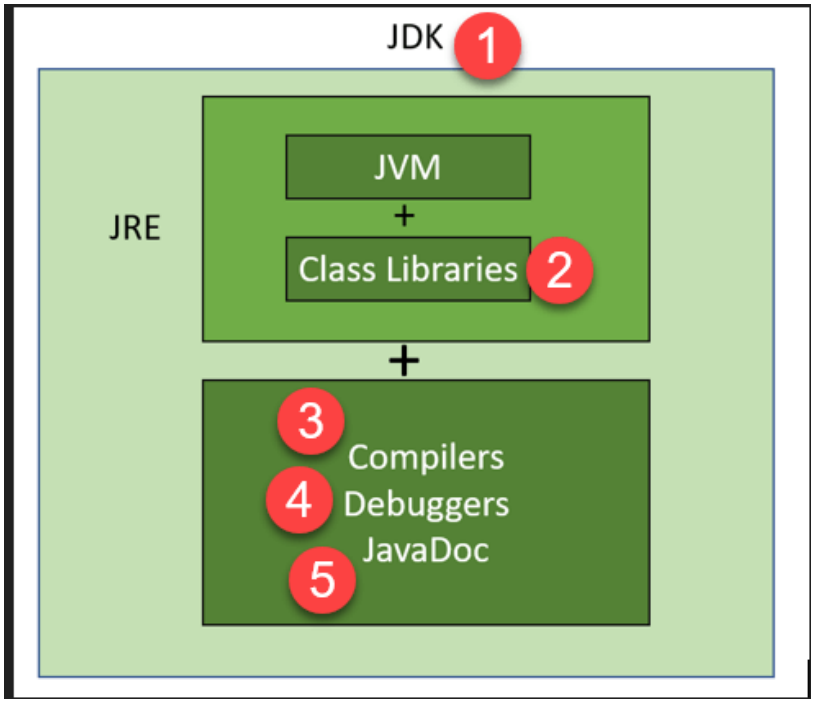
4. System budowania projektu - system pozwalający na automatyzację kompilacji plików źródłowych łączący i kompilujący pliki projektu w celu stworzenia pliku wykonywalnego. W przypadku niniejszego projektu należy wybrać Maven.

5. JDK (ang. Java Deleopment Kit) - Środowisko programistyczne do tworzenia aplikacji w Javie, zawierające takie narzędzia jak kompilator, debugger, środowisko uruchomieniowe Javy (ang. JRE - Java Runtime Environment) i wirtualną maszynę Javy (ang. JVM - Java Virtual Machine). Koncepcja JDK jest przedstawiona na Rysunku 1
W przypadku niniejszego projektu należy wybrać `Oracle OpenJDK`.

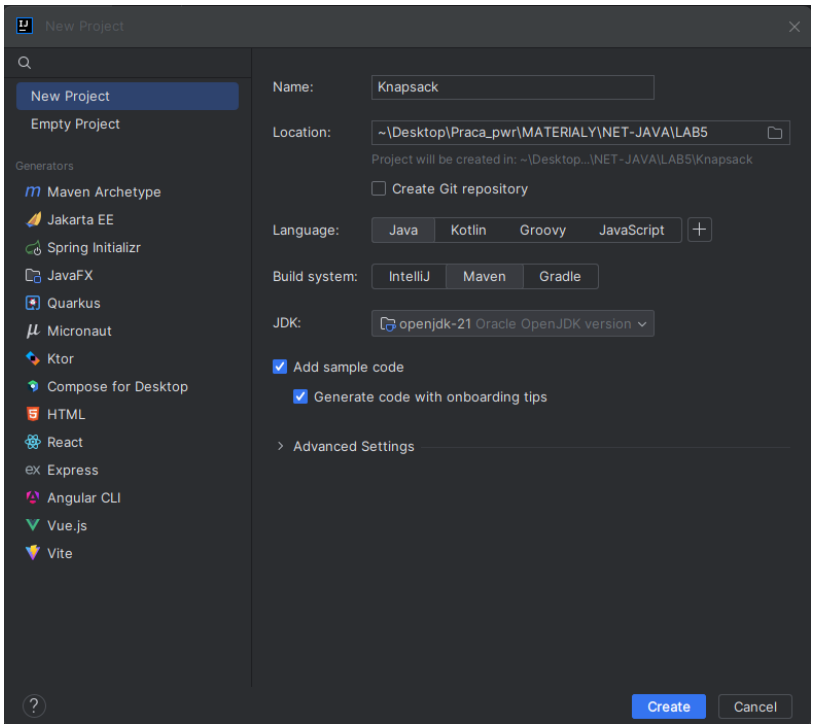
Język programowania `Java` podobnie jak język `C#` jest językiem czysto obiektowym. Funkcja `main` musi być statyczna, publiczna oraz otoczona klasą `Main`.

```
package firstProgram;
public class Main {
    public static void main(String[] args) {
        System.out.print("Wake the f*** Up Samurai, we have a city to burn!");
    }
}
```

Warto umieszczać klasy wewnątrz paczek (pakietów), ponieważ wewnątrz klasy mamy dostęp do innych klas z tej samej paczki.



Rysunek 1: Struktura komponentów JDK



Rysunek 2: Okno tworzenia nowego projektu

Nieograniczony problem plecakowy

W pierwszej kolejności należy przygotować generator instancji nieograniczonego problemu plecakowego zgodnie z jego opisem. W nieograniczonym problemie plecakowym mamy zbiór rodzajów przedmiotów $\mathcal{I} = \{1, 2, \ldots, n\}$, gdzie n to liczba rodzajów. Liczba przedmiotów każdego rodzaju nie jest limitowana, tak jak miało to miejsce w przypadku binarnego problemu plecakowego. Każdy przedmiot jest opisany dwoma parametrami:

- v_i – wartość,
- w_i – waga.

Dla każdego przedmiotu $i \in \mathcal{I}$ należy określić wartość $x_i \in N$, która w przypadku nieograniczonego problemu plecakowego powinna być liczbą naturalną. Pojemność plecaka jest ograniczona wartością C, gdzie suma wag przedmiotów w plecaku nie może przekraczać maksymalnej pojemności plecaka:

$$\sum_{i=1}^{n} x_i w_i \leqslant C. \tag{1}$$

Zadaniem jest znalezienie zbioru przedmiotów o jak największej wartości:

$$\textbf{maximize} \sum_{i=1}^{n} x_i v_i. \tag{2}$$

pamiętając o ograniczeniu (1).

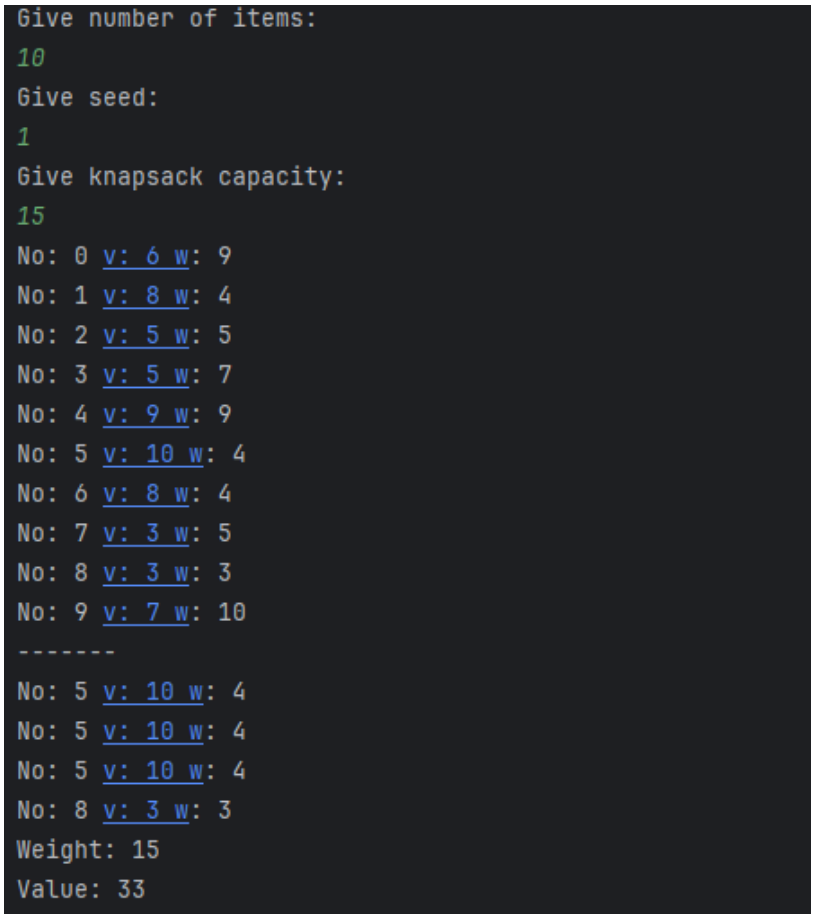
3.2 Sposób implementacji

Aby zaimplementować opisany wcześniej nieograniczony problem plecakowy, należy utworzyć w projekcie nową klasę `Problem`. Klasa ta powinna znaleźć się w jednej paczce Javy, razem z klasą `Main`. Paczka jest to zbiór pozwalający na grupowanie ze sobą podobnych klas, podpakietów i interfejsów, ułatwiający ich użycie i lokalizowanie. W klasie `Problem` należy zdefinowiać pola odpowiedzialne za: (1) liczbę rodzajów przedmiotów, (2) ziarno losowania, (3) dolny i górny zakres liczb, z którego losowane będą waga i wartość przedmiotu oraz (4) listę rodzajów przedmiotów (listę obiektów klasy `Przedmiot`). Listę rodzajów przedmiotów można zastąpić dwoma listami lub tablicami, które będą przechowywać informacje o przedmiotach, czyli ich wagę i wartość. Następnie należy utworzyć konstruktor z czterema parametrami: `n`, `seed`, `lowerBound`, `upperBound`, który wygeneruje nam instancję problemu. Należy skorzystać z domyślnego generatora liczb pseudolosowych, tzn. w pierwszej kolejności tworzymy obiekt klasy `Random`, podając jako parametr konstruktora ziarno generatora. W przypadku, gdy ziarno nie zostanie podane wprost w konstruktorze, będzie ono różne przy każdym utworzeniu obiektu. Następnie przy użyciu metody `nextInt()` należy wylosować liczbę całkowitą - w tym przypadku będącą wagą i wartością danego przedmiotu. Jeśli nie podamy argumentów, to wylosowana liczba będzie liczbą całkowitą, z całego przedziału wartości typu Int, co oznacza iż mogą zostać wylosowane ujemne wartości. Chcąc uzyskać liczbę z danego przedziału, należy odpowiednio zwiększyć losowaną wartość przez dodanie do niej dolnej wartości przedziału oraz losować liczbę z podanego zakresu, będącego różnicą przedziałów górnego i dolnego.
Na potrzeby niniejszego projektu należy przyjąć wartość i wagę pojedynczego przedmiotu z zakresu $v_i, w_i \in < 1, 10 >$

Na koniec należy jeszcze przeciążyć metodę `ToString()`, żeby możliwe było zwracanie instancji naszego problemu jako `string`. Do tego celu należy wykorzystać tag `@Override`. Przeciążenie będzie przydatne w późniejszych etapach implementacji aplikacji - znacznie uprości wykonanie testów i przejście na graficzny UI.

Metoda rozwiązania

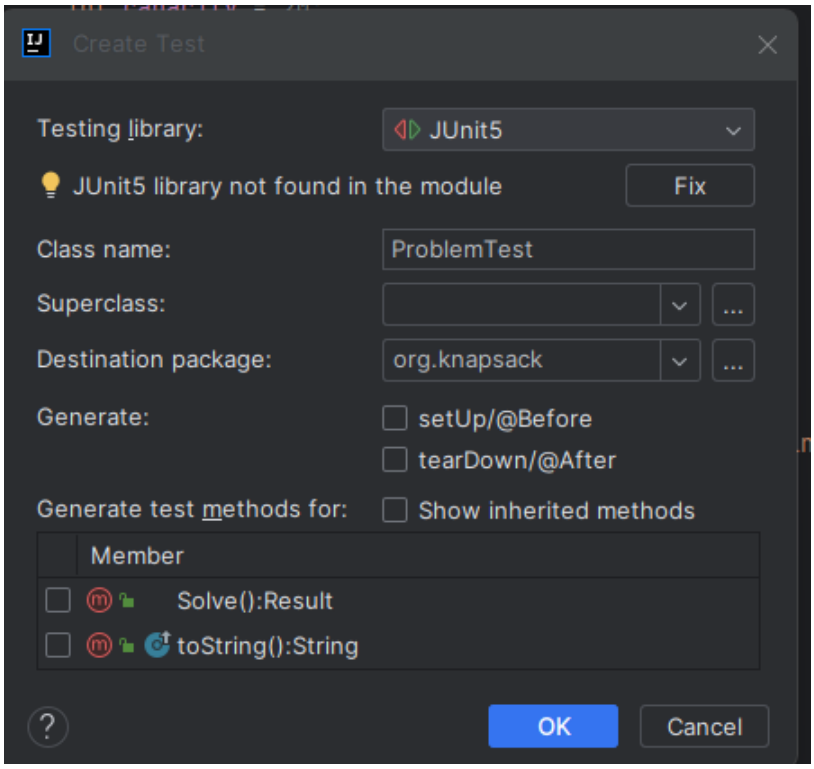
Do rozwiązania problemu należy zaimplementować algorytm aproksymacyjny (Dantzig, 1957). W tym celu trzeba utworzyć metodę `Solve(int capacity)` wewnątrz naszej klasy problemu, która jako argument powinna przyjmować pojemność plecaka. Metoda powinna zwracać rozwiązanie jako obiekt klasy `Result`, zawierający listę numerów przedmiotów w plecaku, ich ilość, sumaryczną wartość, sumaryczną wagę oraz przeciążoną klasę `ToString()`. Wewnątrz metody w pierwszej kolejności należy posortować przedmioty po stosunku ich wartości do wagi. Przedmioty najbardziej opłacalne mają być na początku. Następnie należy kolejno próbować umieszczać przedmioty o największej wartości w plecaku dopóki się w nim mieszczą. Oznacza to, że pierwszy przedmiot o największym stosunku wartości do wagi, powinien być dokładany tak długo aż zabraknie dla niego miejsca. Algorytm zakończy się w momencie sprawdzenia wszystkich przedmiotów lub jeśli plecak zostanie wypełniony całkowicie. Na Rysunku 3 przedstawiono przykładowe działanie programu. Rysunek ma służyć wyłącznie za inspirację.



Rysunek 3: Aplikacja konsolowa – nieograniczony problem plecakowy

4 Zadanie 2

W tym zadaniu należy przećwiczyć tworzenie i przeprowadzanie testów jednostkowych zaimplementowanych funkcjonalności. Do stworzenia testów jednostkowych wykorzystana zostanie bliblioteka `JUnit`, którą należy doinstalować w ramach rozwiązania (można to zrobić automatycznie podczas dodawania nowego testu jednostkowego). Aby utworzyć nowy test jednostkowy do danej klasy, należy otworzyć plik, w którym ona się znajduje, a następnie najechać myszką na jej nazwę i kliknąć prawy przycisk. Następnie z listy rozwijanej wybrać kolejno opcje `GoTo` → Test → `Create new test`. Po wybraniu opisanych opcji pojawi się okno, przedstawione na Rysunku 4, w którym w naszym przypadku należy zdefiniować wersję biblioteki testów, nazwę klasy oraz namespace. Jeśli projekt nie posiada doinstalowanej biblioteki testowej, można wybrać opcję Fix, która doinstaluje wymagane pakiety. Wspomnianą instalację można także przeprowadzić osobno, kilkając prawym przyciskiem myszy na projekt (w drzewie projektu), następnie wybrać opcję `Open module settings` i dodać odpowiedni pakiet w zakładce `Libraries`. Zazwyczaj do każdej klasy powinna być utworzona osobna klasa testów jednostkowym, ale w omawianym przypadku wystarczy jedna klasa. Należy pamiętać o poprzedzeniu każdej metody testowej tagiem `@Test`.



Rysunek 4: Okno tworzenia klasy testowej

W przypadku gdy testy znajdują się w tej samej paczce javy (tym samym namespace), nie ma konieczności dodawania żadnych odwołań do testowanej klasy (odwołanie do pakietu oraz do biblioteki JUnit zostanie dodane automatycznie na początku skryptu). Testy uruchomić można wybierając opcję `Run` pojawiającą się po kliknięciu prawym przyciskiem myszy na klasę testową lub z poziomu górnego paska zadań.

Testy jednostkowe mogą mieć postawiony warunek na różne sposoby, ale zawsze muszą być tak napisane, aby **prawidłowo działający kod przeszedł wszystkie testy**. Na Rysunku 5 zamieszczono przykład testu jednostkowego dla klasy `Problem` generującej `n` obiektów. Test sprawdza czy w rzeczywistości wewnątrz obiektu dodało się do listy dokładnie `n` elementów.

```
    public class ProblemTest {

@Test
    public void NumberCountTest(){
    Problem pr1 = new Problem( num: 5,  sd: 1,  cap: 10,  loB: 10,  upB: 20);
    Assert.assertEquals( expected: 5, pr1.instance.size());

}
```

Rysunek 5: Aplikacja konsolowa – nieograniczony problem plecakowy

Zasadniczym celem zadania jest napisanie czterech, wyszczególnionych poniżej testów jednostkowych, w celu sprawdzenia poprawności działania zaimplementowanych funkcji. Propozycje testów (mogą być inne, o ile zachowany zostanie ich sens):

- Sprawdzenie, czy jeśli co najmniej jeden przedmiot spełnia ograniczenia, to zwrócono co najmniej jeden element.
- Sprawdzenie, czy jeśli żaden przedmiot nie spełnia ograniczeń, to zwrócono puste rozwiązanie.
- Sprawdzenie, czy waga i wartość wszystkich przedmiotów z listy mieści się w założonym przedziale.
- Sprawdzenie poprawności wyniku (sumy wag i wartości w plecaku) dla konkretnej instancji.

W podanych metodach testowych należy wykorzystać różne typy asercji dostępne w ramach frameworku `JUnit`. Istotne jest, aby samodzielnie wymyślone testy były zróżnicowane i miały różnie postawiony warunek.

5 Zadanie 3

Zadanie trzecie polegać będzie na wykonaniu graficznego interfejsu użytkownika z wykorzystaniem biblioteki `Swing`. Środowisko `IntelliJ` podobnie jak `Visual Studio` udostępnia wizard pozwalający na dodawanie kontrolek do okna metodą drag & drop. Oczywiście, samo GUI może być również stworzone bez użycia wizardu, z użyciem odpowiednich komend. Aby rozpocząć pracę z GUI, należy w pierwszej kolejności w ustawieniach programu aktywować UI designera wchodząc kolejno w `File` → `Settings` → `Plugins` i wyszukując `UI Designer`. Następnie, po restarcie IDE, należy pozwolić na automatyczne generowanie klas Javy dla UI designera. W tym celu wchodzimy w `File` → `Settings` → `Editor` → `Editor` i zmieniamy opcję `Generate GUI into` na `Java source code`. Mając poprawnie skonfigurowane środowisko, można dodać do naszego projektu klasę odpowiedzialną za interfejs graficzny (nie ma potrzeby tworzenia nowego projektu - wykorzystujemy wcześniej utworzony do aplikacji konsolowej). W tym celu klikamy prawym przyciskiem myszy na paczkę, w której ma zostać utworzona klasa i wybieramy `New` → `GUI form`. Utworzony zostanie katalog zawierający klasę z kodem naszego interfesju oraz plik projektanta obiektu. W klasie z kodem automatycznie dodane zostaną następujące pakiety, które w przypadku tworzenia GUI bez projektanta należy dołączyć ręcznie:

```
import javax.swing.*;
import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

Elementy możemy dodawać bezpośrednio do okienka lub najpierw grupować w panelach. Elementy wewnątrz `JPanel` będą ustawione w rzędzie w zależności od kolejności dodawania metodą `add(object)`. Ważne, żeby jako `Layout manager` panelu ustawić `GridBagLayout`, który pozwoli na grupowanie elementów na siatce. Mimo, że początkowo wydaje się być to trudniejsze podejście niż swobodne przmieszczanie komponentów, to tak naprawdę znacznie porządkuje interfejs i ułatwia jego implementację. Najczęściej używane komponenty to:

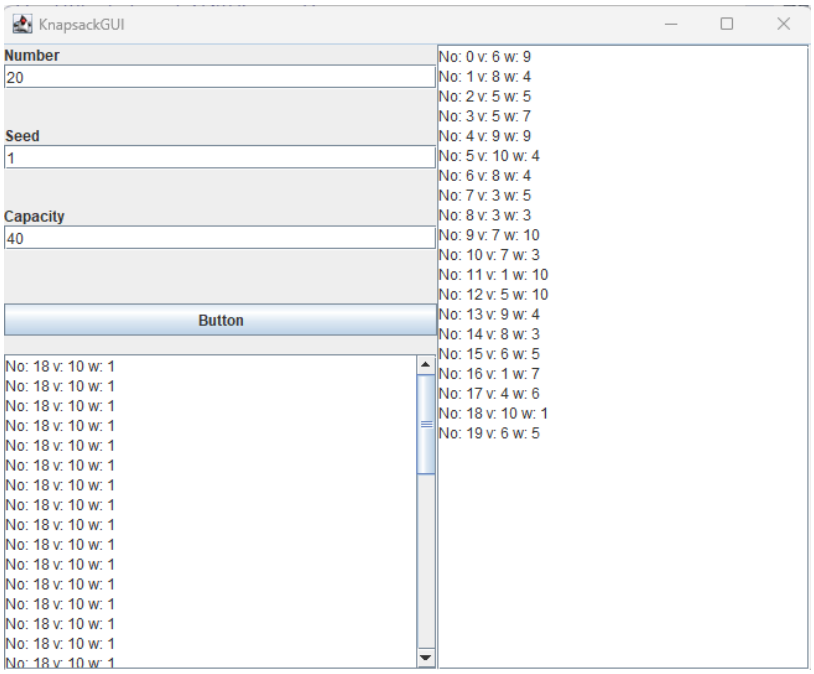
- `JTextField`–pole tekstowe (jeden wiersz),
- `JLabel`–etykieta,
- `JTextArea`–pole tekstowe (wiele wierszy),

- `JButton`–przycisk.

Poniżej wypisano przykład utworzenia elementu o typie `TextArea` i dodanie go do panelu w kodzie klasy GUI:

```
JTextArea textArea = new JTextArea();
textArea.setEditable(false);
textArea.append("long text");
JPanel panel = new JPanel();
panel.add(textArea);
frame.getContentPane().add(BorderLayout.CENTER, panel);
```

Z tworzeniem aplikacji okienkowych w Javie, wiąże się również kilka istotnych faktów, których znajomość jest niezbędna do utworzenia GUI podobnego do przedstawionego na Rysunku 6.



Rysunek 6: Aplikacja konsolowa – nieograniczony problem plecakowy

- W celu dodatnia komponentu należy utworzyć jego obiekt a następnie dodać do panelu lub bezpośrednio do ramki. W przypadku korzystania designera, taki komponent wystarczy przeciągnąć do ramki, nadać mu odpowiednią nazwę oraz zdefiniować wagi (ang. Weight X, Weight Y), które pozwolą na odpowiedni podział siatki pod komponenty (ilość miejsca zajmowanego przez komponent będzie proporcjonalna do jego wagi).
- Jeśli chcemy aby dany komponent zajmował kilka linii/kolumn siatki, należy odpowiednio zdefiniować parametery `gridheight, gridwidth` lub w przypadku designera rozciągnąć komponent na pożądany obszar.
- Chcąc dodać możliwość przewijania zawrtości komponentu w przypadku wygenerowania łańcucha tekstowego z instancją/rozwiązaniem problemu dłuższego niż pojemność komponentu, należy opakować go w kontrolkę `JScrollPane` używając opcji `Surround with...` dostępnej po kliknięciu prawego przycisku myszy.

- Należy zdefiniować, które pola tekstowe będą edytowalne a które nie, ustawiając odpowiedni stan parametru `editable`.
- Dodanie akcji do przycisku odbywa się poprzez dodanie listenera. W naszym przypadku należy wybrać `ActionListener` używając opcji `Create listener` dostępnej po kliknięciu prawego przycisku myszy.

Aby uruchomić aplikację należy utworzyć metodę `main` w klasie naszego GUI. W tym celu można posłużyć się skrótem polegającym na wpisaniu akronimu `psvm` i kliknięciu enter, tworzącego metodę main. W tej metodzie należy następnie utworzyć obiekt okna poprzez wywołanie jego konstruktora oraz zainicjować jego parametry takie jak preferowana wielkość, zachowanie aplikacji po zamknięciu okna czy jego widoczność.

```
JFrame frame = new JFrame("KnapsackGUI");
frame.setContentPane(new KnapsackGUI().panel1);
frame.setPreferredSize(new Dimension(400, 300));
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```