



OPIS ZADANIA

Celem zadania jest Implementacja z wykorzystaniem technologii EJB komponentu o nazwie `SIntegral`. Udostępniona poprzez interfejs o nazwie `ISIntegralRemote` metoda `solve` komponentu `SIntegral` otrzymuje jako parametry liczbę naturalną n oraz stałe a i b . Należy obliczyć oraz zwrócić za pośrednictwem metody `solve(a,b,n)` wyznaczoną wartość całki z dokładnością nie mniejszą niż 10^{-n} .

OPIS MECHANIZMU WYSZUKIWANIA

Do zlokalizowania klasy `FunctMonitor` oraz nawiązania z nią połączenia wykorzystany został interfejs JNDI (ang. Java Naming and Directory Interface). Pozwala on klientom na odkrywanie oraz wyszukiwania danych i obiektów na podstawie nazw.

Aby nawiązać połączenie należy utworzyć obiekt `InitialContext`:

```
InitialContext ctx = new InitialContext();
```

Następnie należy skorzystać z metody `lookup`, której implementacja umożliwia odnalezienie zasobu po nazwie:

```
IFunctMonitor obj = (IFunctMonitor) ctx.lookup("java:global/ejb-project/FunctMonitor!pl.jrj.fnc.IFunctMonitor");
```

Ścieżka przestrzeni nazw ma postać:

```
java:global/ejb-project/FunctMonitor!pl.jrj.fnc.IFunctMonitor
```

ZASTOSOWANY ALGORYTM

Zastosowany algorytm ma za zadanie wyznaczyć wartość poniższej całki, z dokładnością nie mniejszą niż 10^{-n}

$$I = \int_0^a \int_0^b f(x,y) dx dy$$

Jako dane wejściowe otrzymujemy parametry a , b , które kolejno są górnymi granicami obszaru całkowania, oraz parametr n , który wykorzystany zostanie do uzyskania zadanej dokładności. Wartości funkcji $f(x, y)$ jest zwracana w trakcie działania algorytmu.



Aby wyznaczyć wartość zadanej całki skorzystać możemy ze zmodyfikowanej metody trapezów. Dzielimy kolejno oś x oraz y na n segmentów. Kolejno wyznaczamy długość pojedynczego segmentu dla osi x oraz dla osi y . Następnie iterując po ilości segmentów wyznaczamy wysokość, która jest średnią wartością narożników bieżącego segmentu. W każdej iteracji wyznaczamy objętość jako iloczyn wysokości oraz długości i ilości segmentów.

Końcowy rezultat zapisać możemy jako sumę objętości iloczynów długości oraz ilości segmentów x i y .

Poniżej przedstawiony został przykładowy wynik algorytmu dla zadanej funkcji:

$$\int_0^3 \int_0^6 (2x + y^2) dy dx = 270$$

Wynik zwrócony przez aplikację: 270.00010800000086

STRUKTURY DANYCH

W skład projektu wchodzi klasy `SIntegral.java`, `Solver.java` oraz interfejsy `ISIntegralRemote.java`, `IFunctMonitor.java`.

Poniżej przedstawione zostały najważniejsze struktury:

Klasa `Solver.java`

```
void doGet(HttpServletRequest req, HttpServletResponse res)
void printResult(PrintWriter writer)
void prepareParameters(HttpServletRequest request)
```

Klasa `Solver` realizuje funkcję Servletu, w metodzie `doGet` zaimplementowana została logika pobrania parametrów oraz wywołania metody `solve`. Metoda `prepareParameters` odpowiada za przygotowanie parametrów. Metoda `printResult` odpowiada za wyświetlenie rezultatu.

Klasa `SIntegral.java`

```
double solve(double a, double b, int n)
void calculateIntegralValue(IFunctMonitor m, double a, double b, int n)
IFunctMonitor getBean()
double getResult()
```

Klasa `SIntegral` odpowiedzialna jest za realizację głównego algorytmu oraz zwrócenie wyniku obliczeń za pośrednictwem metody `solve`. Metoda `getBean` odpowiedzialna jest za pobranie obiektu `FunctMonitor`. Metoda `calculateIntegralValue` odpowiedzialna jest za obliczenie wartości całki w zadanym obszarze.



Interfejs **ISIntegralRemote.java**

```
double solve(double a, double b, int n)
```

Interfejs **ISIntegralRemote** udostępnia metodę zwracającą wyniki obliczeń.

Interfejs **IFunctMonitor.java**

```
double f( double x, double y );
```

Interfejs **IFunctMonitor** udostępnia metodę **f** zwracającą wartość pewnej funkcji dwóch zmiennych.

URUCHOMIENIE

Proces kompilacji jest możliwy z użyciem komendy:

```
javac -extdirs <path-to-appserver>/lib -Xlint SIntegral.java  
ISIntegralRemote.java Solver.java IFunctMonitor.java
```

Uruchomienie programu możliwe jest w środowisku serwera aplikacyjnego GlassFish 4.