Programowanie obiektowe

Wykład 14: Typ wyliczeniowy. Przykład. Podsumowanie.

Po co typ wyliczeniowy?

- Zbiór wartości określony przez użytkownika
- Symboliczne nazwy wartości
- Automatyczna kontrola

Przykład 1

public enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY }

```
public class EnumTest {
Day day;
public EnumTest(Day day) {
    this.day = day; }
public void tellItLikeItIs() {
  switch (day) {
    case MONDAY:
      System.out.println("Mondays are bad.");
      break;
    case FRIDAY:
     System.out.println("Fridays are better.");
      break;
    case SATURDAY:
    case SUNDAY:
       System.out.println("Weekends are
        best."); break;
    default:
      System.out.println("Midweek days are so-so.");
break;
}
}
```

```
public static void main(String[] args) {
EnumTest firstDay = new EnumTest(Day.MONDAY);
firstDay.tellItLikeItIs();
EnumTest thirdDay = new numTest(Day.WEDNESDAY);
thirdDay.tellItLikeItIs();
EnumTest fifthDay = new EnumTest(Day.FRIDAY);
fifthDay.tellItLikeItIs();
EnumTest sixthDay = new EnumTest(Day.SATURDAY);
sixthDay.tellItLikeItIs();
EnumTest seventhDay = new EnumTest(Day.SUNDAY);
seventhDay.tellItLikeItIs();
 }
}
```


Typ wyliczeniowy jako klasa

- java.lang.Enum
- Metody: valueOf(), name(), ordinal(), itp.

Np.

```
Day d;
d=new Day(MONDAY);
d=SUNDAY;
```

Przykład 2

```
public enum Planet {
  MERCURY (3.303e+23, 2.4397e6),
  VENUS (4.869e+24, 6.0518e6),
  EARTH (5.976e+24, 6.37814e6),
  MARS (6.421e+23, 3.3972e6),
  JUPITER (1.9e+27, 7.1492e7),
  SATURN (5.688e+26, 6.0268e7),
  URANUS (8.686e+25, 2.5559e7),
  NEPTUNE (1.024e+26, 2.4746e7);
 private final double mass; // in kilograms
 private final double radius; // in meters
 Planet(double mass, double radius) {
   this.mass = mass;
   this.radius = radius;
  }
 private double mass() { return mass; }
 private double radius() { return radius; }
 public static final double G = 6.67300E-11;
 double surfaceGravity() {
     return G * mass / (radius * radius);
 }
 double surfaceWeight(double otherMass) {
     return otherMass * surfaceGravity();
 }
```

```
public static void main(String[] args) {
  double earthWeight = Double.parseDouble(args[0]);
  double mass = earthWeight/EARTH.surfaceGravity();
  for (Planet p : Planet.values())
     System.out.printf("Your weight on %s is %f%n",
       p, p.surfaceWeight(mass));
  }
}
```


Kolekcje dla typu wyliczeniowego

- EnumSet – zawiera statyczne metody:
  - range(E from, E to),
  - of(E first, E… last),
  - complementOf(), copyOf(), allOf()
- EnumMap – klucze mogą być tylko wartościami typu wyliczeniowego:
  - size(), containsValue(), containsKey(), get(), put(), remove(), putAll(), clear(), keySet(), values(), entrySet(),

Przykład 3

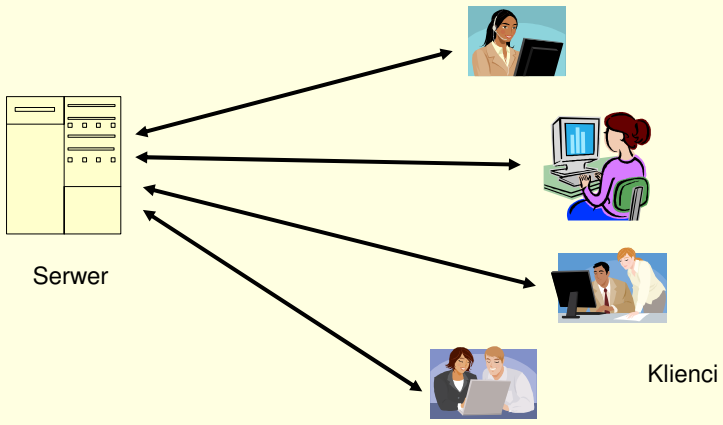
```
for (Day d : EnumSet.range(Day.MONDAY, Day.FRIDAY))
        System.out.println(d);


public enum Operation {
        PLUS, MINUS, TIMES, DIVIDE;
        double eval(double x, double y){
        switch(this) {
        case PLUS: return x + y;
        case MINUS: return x - y;
        case TIMES: return x * y;
        case DIVIDE: return x / y;
        } throw new AssertionError("Unknown op: " + this); }
}
```


Pytania

1. Cechy typu wyliczeniowego.
2. Czy można usunąć/zmienić wartości typu wyliczeniowego?

Przykład: Czat internetowy



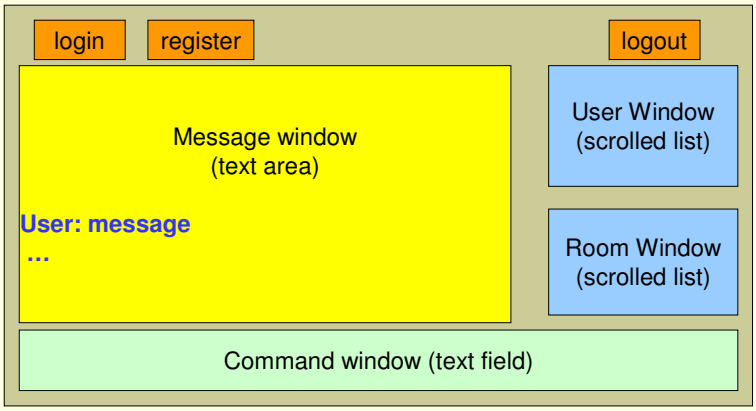
Funkcje serwera

- Obsługa połączeń z aplikacjami klienckimi
- Zarządzanie użytkownikami
  - Logowanie
  - Rejestracja
  - Hasła
  - Baza danych zarejestrowanych użytkowników
  - Baza danych stałych pokoi
- Obsługa komunikatów
  - Wiadomości prywatne
  - Wiadomości publiczne
  - Informacje o statusie użytkowników
- Przetwarzanie poleceń
  - Odebranie polecenia od aplikacji klienckiej
  - Przetworzenie polecenia
  - Wysłanie odpowiedzi

Funkcje aplikacji klienckiej

- Obsługa użytkownika
  - Użytkownicy stali (wymagana rejestracja i hasło)
  - Użytkownicy tymczasowi (dowolna unikalna nazwa)
- Intrerfejs GUI
  - Okno komunikatów (MessageWindow)
  - Okno poleceń (CommandWindow)
  - Okno z listą użytkowników (UsersWindow)
  - Okno z listą pokoi (RoomWindow)
- Komunikacja z serwerem
  - Wysyłanie komend do serwera
  - Odbiór odpowiedzi
  - Odbiór i przetwarzanie komend z serwera

GUI aplikacji klienckiej



Polecenia aplikacji klienckiej

- @register *user password*
- @login *user password*
- @logout *user*
- @color *text_color*
- @priv *user message*
- @busy
- @join *room*
- @create *room*
- @remove *room*
- *message*

Polecenia serwera

- $adduser *user room*
- $removeuser *user room*
- $setbusy *user*
- $setavail *user*
- $display *message user color*
- $accept *user*
- $deny *user*
- $addroom *room*
- $removeroom *room*

Akcje serwera

- Start aplikacji (main()):
  - Utworzenie instancji klasy Server
  - Wczytanie bazy zarejestrowanych użytkowników: loadUsers()
  - Wczytanie bazy pokoi, a gdy lista pusta utworzenie pokoju publicznego „towarzyski”: loadRooms()
  - Inicjalizacja listy klientów (pusta)
  - Uruchomienie pętli oczekiwania na klientów (mainLoop())
- Zgłoszenie aplikacji klienckiej:
  - Utworzenie wątku UserThread
  - Dodanie wątku do clientList

Obsługa klienta: ClientThread(1)

- Inicjalizacja (run()):
  - Utworzenie strumieni we/wy
  - Uruchomienie pętli:
    - oczekiwanie na następny komunikat od klienta getMessage()
    - przetworzenie komunikatu: processMessage()
    - wysłanie odpowiedzi: sendMessage()

Przetwarzanie komunikatów klienta (1)

- @register user password
  - Dodanie użytkownika do bazy: **regUser()**
  - Wysłanie odpowiedzi:
    $accept user – gdy OK, lub $deny user – gdy niepowodzenie
- @login user password
  - Gdy password=„”:
    - Utworzenie użytkownika tymczasowego
    - Wysłanie odpowiedzi:
      $accept user – gdy OK, lub $deny user – gdy niepowodzenie
  - Gdy password<>„”:
    - Sprawdzenie użytkownika w bazie
    - Wysłanie odpowiedzi:
      $accept user – gdy OK, lub $deny user – gdy niepowodzenie
  - Gdy OK:
    - Dołączenie użytkownika do pokoju Towarzyski
    - Wysłanie do klienta:
      $addroom room – dla wszystkich pokoi
      $adduser user room - dla wszystkich zalogowanych użytkowników
    - Wysłanie do wszystkich:
      $adduser user Towarzyski

Przetwarzanie komunikatów klienta (2)

- @logout user
  - Wysłanie do wszystkich:
    $removeuser user room
  - Usunięcie klienta z listy clientList
- @color text_color
  - Modyfikacja statusu klienta
- @priv *user message*
  - Wysłanie do klienta zalogowanego jako *user*
    $display message client.user.name client.color
- @busy
  - Modyfikacja statusu klienta
  - Wysłanie do wszystkich:
    $setbusy user

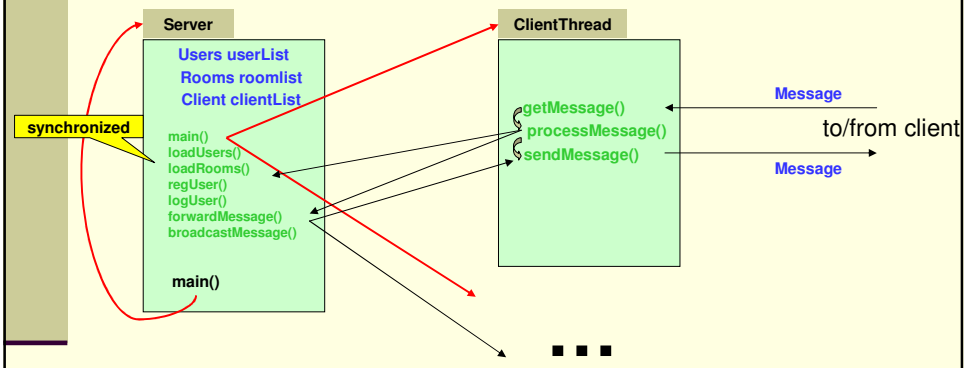
Przetwarzanie komunikatów klienta (3)

- @join *room*
  - Wysłanie do wszystkich:
    $removeuser user client.room
  - Dodanie klienta do pokoju room
  - Wysłanie do wszystkich:
    $adduser user room
- @create *room*
  - Utworzenie pokoju administrowanego przez klienta
  - Wysłanie do wszystkich:
    $addroom room
- @remove *room*
  - Usunięcie pokoju
  - Wysłanie do wszystkich (jeśli OK.):
    $removeroom room

Przetwarzanie komunikatów klienta (4)

- Komunikat
  - Wysłanie do wszystkich:
    $display komunikat client.user client.color
  - jeśli klient był niekatywny:
    - Zmiana statusu klienta
    - Wysłanie do wszystkich:
      $setavail user
- Po zerwaniu połączenia:
  - Zamknięcie połączenia
  - Wysłanie do wszystkich:
    $removeuser user room
  - Usunięcie wątku klienta

Projekt aplikacji serwera



Główne klasy(1)

- Server
  - Pola:
    - private Users userList
    - Rooms roomList
    - Clients clientList
  - Metody:
    - static public main(String[] params)
    - private Users loadUserlist(String plik)
    - private Rooms loadRooms(String plik)
    - public boolean regUser(String nazwa, String haslo)
    - public boolean logUser(String nazwa, String haslo)
    - public void broadcastMessage(String message)
    - public void forwardMessage(String Message, String username)

Główne klasy (2)

- ClientThread
  - Pola:
    - private Socket s
    - private ObjectInputStream inp
    - private ObjectOutputStream outp
    - private ClientMessage messageIn
    - private ServerMessage messageOut
    - private Server serv
    - private Client client
  - Metody:
    - public void run()
    - private ClientMessage getMessage()
    - private ServerMessage processMessage(ClientMessage m)
    - public synchronized void sendMessage(ServerMessage m)

Główne klasy (3)

- abstract class Message
  - Pola:
    - String[] params
  - Methods:
    - public String getParam(int i)
    - abstract Message process(Object app)
- abstract class ServerMessage:
  - Klasy pochodne: MAddRoom,MAddUser, MDelUser, MDelRoom, MSetBusy, MSetAvail, MAccept, MDeny, MDisplay
- abstract class ClientMessage:
  - Klasy pochodne: MRegister, MLogin, MLogout, MBusy, MColor, MPriv, MMess, MCreateR, MRemoveR, MJoinR

Główne klasy (4)

- class MRegister extends ClientMessage
  - Konstruktor: MRegister(String name, String haslo)
  - Metody:
    - serverMessage process(Object app):
      - s = (Server) app;
      - if (s.regUser(name,haslo))
        return new MAccept(name) else return new MDeny(name)

Wtedy metoda z klasy ClientThread:

```
serverMessage processMessage(m){
        return m.process(serv)
}
```


Główne klasy (5)

- Client – dane o zalogowanym kliencie
  - Pola:
    - private ClientThread ct
    - private User user
    - Room room
    - boolean busy
    - int color
  - Metody:
    - public User getUser()
    - public removeClient()

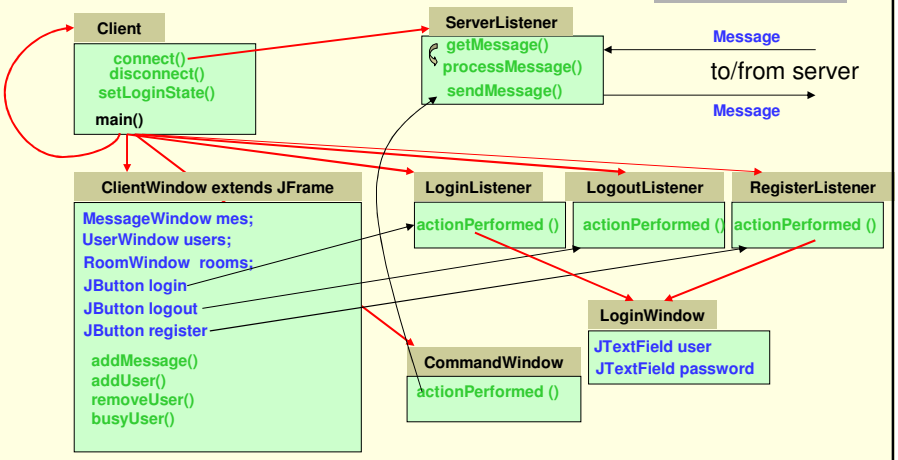
Główne klasy (6)

- User
  - Pola:
    - private String name
    - private String password
  - Metody:
    - public boolean checkPass(String pass)
    - public String getName()
- Users
  - Pola
    - private LinkedList userlist
  - Metody:
    - public boolean addUser(User u)
    - public boolean removeUser(String name)

Główne klasy (7)

- Room
  - Pola:
    - private String nazwa
    - private TreeSet<User> userlist
    - private User admin
  - Metody:
    - public User getAdmin()
    - public String getName()
    - public boolean addUser(User u)
    - public boolean removeUser(User u)
    - public TreeSet<User> getUserList()
- Rooms
  - Pola
    - private LinkedList<Room> roomlist
  - Metody:
    - public boolean addRoom(Room r)
    - public boolean removeRoom(String name)
    - public boolean exists(String name)

Aplikacja kliencka



Przetwarzanie poleceń serwera

- $adduser user
  addUser() - userWindow.addElement(user)
- $remove user
  removeUser() - userWindow.removeElement(user)
- $setbusy user
  busyUser(true) - userWindow.busyElement(user),
  e.g. available users - user_name
  busy users - #user_name
- $setavail user
  busyUser(false) - userWindow.availElement(user)
- $display message user color
  addMessage() - messageWindow.insertLine(message,color)
- $accept user
  setLoginState(true)
- $deny user
  setLoginState(false)

Inne funkcje

- Obsługa zerwanych połączeń
- Zarządzanie użytkownikami:
  - Usuwanie nieaktywnych, zarejestrowanych użytkowników
  - Usuwanie użytkowników naruszających zasady czatu
- Zarządzanie pokojami:
  - Usuwanie pokoi
  - Kontrola dostępu do pokoi
  - Pokoje stałe i tymczasowe

Praktyka programowania obiektowego (1)

- Śpiesz się powoli (elegancja kodu)
- Niepotrzebnie nie komplikuj kodu
- „Dziel i rządź”
- Separacja twórcy od użytkownika klasy
- Używaj nazw nie wymagających komentarza
- Twórz kod testujący przed implementacją klasy
- Rozpocznij projekt od określenia klas, ich publicznych interfejsów i wzajemnych relacji
- Klasy powinny być jak najprostsze

Praktyka programowania obiektowego (2)

- Unikaj długich list argumentów
- Unikaj powtarzania kodu
- Nie nadużywaj instrukcji switch oraz if-else (może lepiej dziedziczenie lub polimorfizm?)
- Podstawowe funkcje powinny być w klasach bazowych
- Łatwiej jest rozszerzać klasy niż „zawężać” (nie umieszczaj w klasie wszystkich możliwych metod)
- Stosuj dziedziczenie jeśli będzie potrzeba rzutowania w górę (w przeciwnym przypadku rozważ kompozycję)

Praktyka programowania obiektowego (3)

- Jeśli metody zachowują się zależnie od stanu zmiennych (albo od parametrów), rozważ reorganizację kodu (np. przeciążanie nazw)
- Stosuj hierarchię wyjątków
- Twórz kod dla innych (czytelny)
- Unikaj „gigantycznych” obiektów (nawyk z programowania proceduralnego)
- Obiekty nie powinny składać się z samych pól
- Korzystaj z mechanizmów programowania obiektowego

Koniec