Wątki

Streszczenie

Celem wykładu jest wprowadzenie do obsługi wątków w Javie.
Czas wykładu – 45 minut.

Definiowanie wątków jako klas potomnych Thread

Nadpisanie metody `run()`.

```
class Watek extends Thread
{
    public void run()
    {
        try
        {
            for (int i = 0; i < 5; i++)
            {
                System.out.println(getName()+":"+i);
                sleep(1);
            }
        } catch (InterruptedException e)
        {
        }
    }
}

class watki {
    public static void main (String args[])
    {
        ( new Watek() ) . start();
        ( new Watek() ) . start();
        ( new Watek() ) . start();
    }
}
```

Definiowanie wątków jako klas implementujących Runnable

Implementacja metody `run()`.

```
class Watek implements Runnable
{
    public void run()
    {
        try {
            for (int i = 0; i < 5; i++)
            {
                System.out.println(i);
                Thread.sleep(1);
            }
        } catch (InterruptedException e)
        {
        }
    }
}

class watki {
    public static void main (String args[])
    {
        ( new Thread( new Watek() ) ) .start();
        ( new Thread( new Watek() ) ) .start();
        ( new Thread( new Watek() ) ) .start();
    }
}
```

Synchronizacja wątków — funkcje

```
class Klasa {
    synchronized // usunąć
        void f (String s) { // metoda wypisuje nawias przed i po napisie
        System.out.print("f("+s);
        try {
            Thread.sleep(1);
        } catch (InterruptedException e) {
        }
        System.out.println(")");
    }
}

class Watek implements Runnable {
    Thread t = new Thread(this);
    Klasa k;
    Watek(Klasa a) {
        k = a;
    }
    public void run() {
        k.f(t.getName());
    }
}

class watki {
    public static void main (String args[]) {
        Klasa
            o1 = new Klasa(),
            o2 = new Klasa();
        Watek
            w1 = new Watek(o1),
            w2 = new Watek(o1), // przetestować o2
            w3 = new Watek(o1);
        w1.t.start();
        w2.t.start();
        w3.t.start();
    }
}
```

Synchronizacja wątków — blok

```
class Klasa {
    void f(String s) { // metoda wypisuje nawias przed i po napisie
        System.out.print("f("+s);
        try {
            Thread.sleep(1);
        } catch (InterruptedException e) {
        }
        System.out.println(")");
    }
}

class Watek implements Runnable {
    Thread t = new Thread(this);
    Klasa k;
    Watek(Klasa a) {
        k = a;
    }
    public void run() {
        synchronized (k) // obiekt, dla którego wołamy metodę
            { // koniecznie blok
                k.f(t.getName());
            }
    }
}

class watki {
    public static void main (String args[]) {
        Klasa
            o1 = new Klasa(),
            o2 = new Klasa();
        Watek
            w1 = new Watek(o1),
            w2 = new Watek(o1), // przetestować o2
            w3 = new Watek(o1);
        w1.t.start();
        w2.t.start();
        w3.t.start();
    }
}
```

Synchronizacja wątków — semafor

```
class Watek implements Runnable {
    Thread t = new Thread(this);
    static Object SEMAFOR = new Object();
    void pisz() {
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println(t.getName()+":"+i);
                Thread.sleep(1);
            }
        } catch (InterruptedException e) {
        }
    }
    public void run() {
        synchronized (SEMAFOR)
            {
                pisz();
            }
    }
}

class watki {
    public static void main (String args[]) {
        ( new Watek() ) .t.start();
        ( new Watek() ) .t.start();
        ( new Watek() ) .t.start();
    }
}
```

Synchronizacja wątków — priorytety

```
class Klasa {
    synchronized
        void f(String s) {
        System.out.print("f("+s);
        try {
            Thread.sleep(1);
        } catch (InterruptedException e) {
        }
        System.out.println(")");
    }
}

class Watek implements Runnable {
    Thread t = new Thread(this);
    Klasa k;
    Watek(Klasa a) {
        k = a;
    }
    public void run() {
        try {
            Thread.sleep(1);
        } catch (InterruptedException e) {
        }
        k.f(t.getName());
    }
}

class watki {
    public static void main (String args[]) {
        Klasa o1 = new Klasa();
        Watek
            w1 = new Watek(o1),
            w2 = new Watek(o1),
            w3 = new Watek(o1);
        w1.t.setPriority(4); // zakres 1-10, default 5
        w3.t.setPriority(6); // zakres 1-10, default 5
        w3.t.setName("Najwazniejszy");
        w1.t.start();
        w2.t.start();
        w3.t.start();
    }
}
```

Zawieszanie wątków — wait

```
class Watek implements Runnable {
    Thread t = new Thread(this);
    public void run() {
        for (int i = 0; i < 5; ++i) {
            try {
                Thread.sleep(1);
                synchronized (this) // wykomentować
                {
                    System.out.println(i);
                    if (i == 3) wait();  // wykomentować
                    //if (i == 3) wait(1000);  // wykomentować
                }
            } catch (InterruptedException e) {
            }
        }
    }
}

class watki {
    public static void main (String args[]) {
        Watek
            w1 = new Watek(),
            w2 = new Watek(),
            w3 = new Watek();
        w1.t.start();
        w2.t.start();
        w3.t.start();
    }
}
```

Kolejkowanie wątków — wait, notify — błędna implementacja

```
class Kolejka {
    int a;
    synchronized void put(int b) {
        a = b;
        System.out.println("Wlozyles " + a);
    }
    synchronized int get() {
        System.out.println("Pobrales " + a);
        return a;
    }
}

class Producent implements Runnable {
    Kolejka k;
    Thread t = new Thread(this,"Producent");
    Producent(Kolejka q) {
        k = q;
    }
    public void run() {
        int i = 0;
        while (true) {
            k.put(++i);
            if (i==5) System.exit(1);
        }
    }
}

class Konsument implements Runnable {
    Kolejka k;
    Thread t = new Thread(this,"Konsument");
    Konsument(Kolejka q) {
        k = q;
    }
    public void run() {
        while (true) {
            k.get();
        }
    }
}

class watki {
    public static void main (String args[]) {
        Kolejka k = new Kolejka();
        (new Producent(k)).t.start();
        (new Konsument(k)).t.start();
    }
}
```

Kolejkowanie wątków — wait, notify — rozwiązanie

```
class Kolejka {
    int a;
    boolean polozony = false;
    synchronized void put(int b) {
        if (polozony)
            try {
                wait();  // usypia wątek i oddaje monitor do momentu
                // zawołania 'notify' przez inny wątek
            } catch (InterruptedException e) {
            }
        a = b;
        polozony = true;
        System.out.println("Wlozyles " + a);
        notify(); // budzi wątek, ktąry wywołał 'wait' dla tego samego obiektu,
        // jeśli takie istnieją  (notifyAll)
    }
    synchronized int get() {
        if (!polozony)
            try {
                wait();
            } catch (InterruptedException e) {
            }
        System.out.println("Pobrales " + a);
        polozony = false;
        notify();
        return a;
    }
}

class Producent implements Runnable {
    Kolejka k;
    Thread t = new Thread(this,"Producent");
    Producent(Kolejka q) {
        k = q;
    }
    public void run() {
        int i = 0;
        while (true) {
            k.put(++i);
            if (i==5) System.exit(1);
        }
    }
}

class Konsument implements Runnable {
    Kolejka k;
```

```
    Thread t = new Thread(this,"Konsument");
    Konsument(Kolejka q) {
        k = q;
    }
    public void run() {
        while (true) {
            k.get();
        }
    }
}

class watki {
    public static void main (String args[]) {
        Kolejka k = new Kolejka();
        (new Producent(k)).t.start();
        (new Konsument(k)).t.start();
    }
}
```