Programowanie obiektowe

Wątki: interfejsy i dziedziczenie

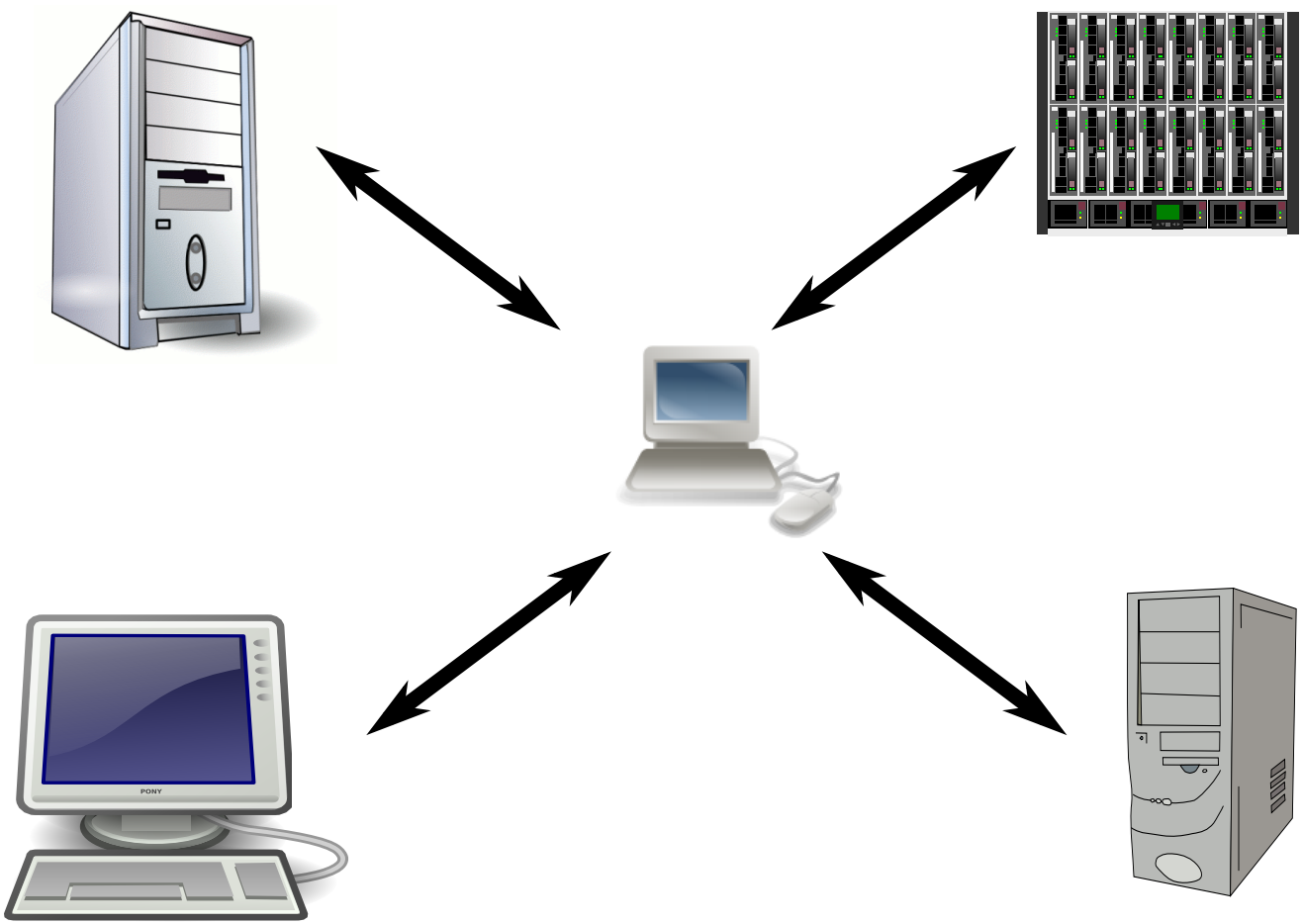
Marcin Młotkowski

11 kwietnia 2019

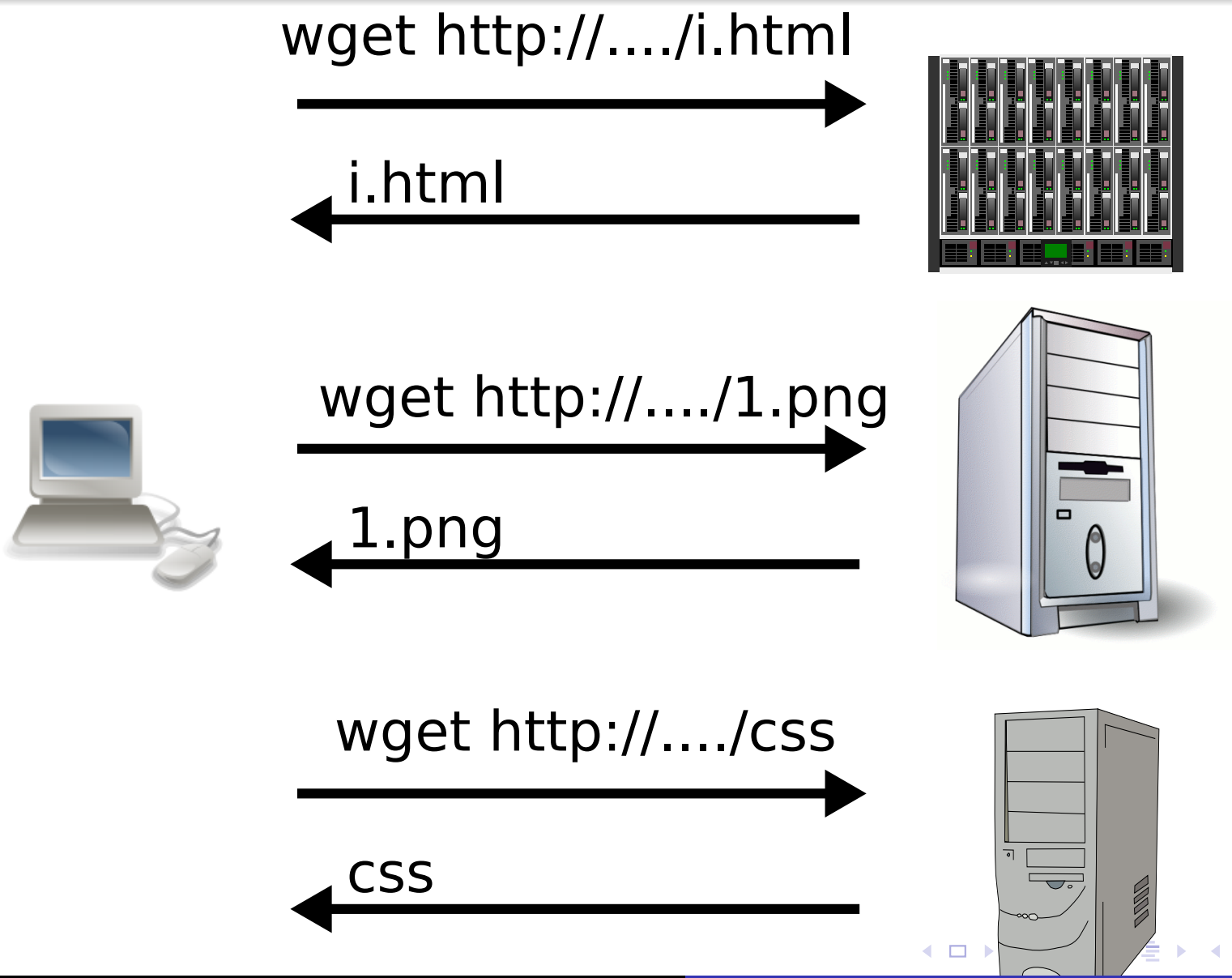
Plan wykładu

Plan wykładu

Motywacje



Scenariusz



Przykład

Wyścigi wątków

- każdy wątek ma wykonać 400 000 razy pustą instrukcję;
- co 50 000 wątek ma się "zgłosić".

Realizacja wątków

```
public class Thread
{
    public void run();
}
```

Realizacja wątków

```
public class Thread
{
    public void run();
}
```

Jak skorzystać:

deklarując podklasę `Thread` i implementując własną metodę `run()`.

Klasa Biegacz

```
class Biegacz extends Thread
{
    private int tick = 1;
    private int num;
    public Biegacz(int num) { this.num = num; }
    public void run() { ... }
}
```

Metoda run()

```
public void run()
{
    while (this.tick < 400000)
    {
        this.tick++;
        if ((this.tick % 50000) == 0)
            System.out.println("Thread #" + num);
    }
}
```

Zawody

```
Biegacz [ ] runners = new Biegacz[NUMRUNNERS];
for (int i = 0; i < NUMRUNNERS; i++)
{
    runners[i] = new Biegacz(i);
}
```

Zawody

```
Biegacz [ ] runners = new Biegacz[NUMRUNNERS];
for (int i = 0; i < NUMRUNNERS; i++)
{
    runners[i] = new Biegacz(i);
}

// START!!!
for (int i = 0; i < NUMRUNNERS; i++)
    runners[i].start();
```

Wynik działania

Thread #0, tick = 50000
Thread #0, tick = 100000
Thread #0, tick = 150000
Thread #1, tick = 50000
Thread #1, tick = 100000
Thread #0, tick = 200000
Thread #1, tick = 150000
Thread #0, tick = 250000
Thread #1, tick = 200000
...
Thread #0, tick = 400000

Thread #1, tick = 50000
Thread #1, tick = 100000
Thread #1, tick = 150000
Thread #1, tick = 200000
Thread #0, tick = 50000
Thread #1, tick = 250000
Thread #0, tick = 100000
Thread #0, tick = 150000
Thread #1, tick = 300000
...
Thread #0, tick = 400000

Dylemat

```
class Ładny_obrazek extends Figura
{
    public void narysuj() { ... }
    public void run()
    {
        this.narysuj();
    }
}
```

Biblioteka standardowa

```
public class Thread
{
    public Thread(Runnable target);
}
```

Biblioteka standardowa

```
public class Thread
{
    public Thread(Runnable target);
}

public interface Runnable
{
    void run();
}
```

Zastosowanie

```
class Biegacze implements Runnable
{
    public void run() { ... }
}
```

Paradoks Zenona z Elei

Achilles i żółw stają na linii startu wyścigu na dowolny, skończony dystans. Achilles potrafi biegać 2 razy szybciej od żółwia i dlatego na starcie pozwala mu się oddalić o 1/2 całego dystansu. Achilles, jako biegnący 2 razy szybciej od żółwia, dobiegnie do 1/2 dystansu w momencie, gdy żółw dobiegnie do 3/4 dystansu.

Wniosek: Achilles nigdy nie dogoni żółwia, mimo że biegnie od niego dwa razy szybciej, gdyż zawsze będzie dzieliła ich zmniejszająca się odległość.

Wspólny interfejs

```
interface Biegacze extends Runnable
{
    void setDistance(int dist);
    int distance();
}
```

Implmentacja

```
class Heros extends BohaterowieMitologii
{
    ...
}

class Żółw extends Kręgowce
{
    ...
}
```

Implmentacja

```
class Heros extends BohaterowieMitologii
implements Biegacze {
    ...
}

class Żółw extends Kręgowce implements Biegacze {
    ...
}
```

Wyścig

```
żółw = new Żółw();
achilles = new Heros("Achilles");
Thread t1 = new Thread(żółw);
Thread t2 = new Thread(achilles);

t1.setPriority(1);
t2.setPriority(2);

t1.start();
while (żółw.distance() < halfDistance);
t2.start();
```

Wady wątków

- czasem mocno obciążają system;
- wymagają starannej implementacji przy korzystaniu ze wspólnych zasobów.

Plan wykładu

Przypomnienie

```
class Stos
{
    public void push(int elem);
    public int pop();
}
```

Scenariusz I

```
Stos stos = new Stos(10);
Wątek w1 = new Wątek(stos);
Wątek w2 = new Wątek(stos);
```

Wątek w1	Wątek w2
`stos.push(10);`	
`this.stos[this.top] = 10;`	
`this.top++;`	
	`stos.push(12);`
	`this.stos[this.top] = 12;`
	`this.top++;`

Scenariusz II

```
Stos stos = new Stos(10);
Wątek w1 = new Wątek(stos);
Wątek w2 = new Wątek(stos);
```

Wątek w1	Wątek w2
stos.push(10);	
	stos.push(12);
this.stos[this.top] = 10;	
	this.stos[this.top] = 12;
this.top++;	
	this.top++;

Struktury danych a wątki

Struktury danych bezpieczne ze względu na wątki (thread–safe)

Struktury danych, które są przystosowane do stosowania w wątkach.

Sekcja krytyczna

Rozwiązanie problemu

Tylko jeden wątek może w danym przedziale czasowym może operować na stosie. Pozostałe wątki muszą czekać aż poprzedni wątek skończy.

Synchronizacja wątków

```
synchronized (obj)
{
    ...
}
```

Działanie instrukcji

- tylko jeden wątek może założyć blokadę na obiekt;
- wątek który nie założy blokady musi czekać na jej zwolnienie.

Implementacja blokad w klasie Stos

```
class Stos
{
    public void push(int elem)
    {
        synchronized (this) { ... }
    }

    public int pop()
    {
        synchronized (this) { ... }
    }
}
```

Trochę lepsza implementacja

```
class Stos
{
    public synchronized void push(int elem)
    {
        ...
    }

    public synchronized int pop()
    {
        ...
    }
}
```

Standardowe struktury danych

Kolekcje dopuszczone do działania w wątkach

`Hashtable`, `Vector`

Klasy dedykowane dla wątków

`ConcurrentHashMap`, `ConcurrentLinkedQueue<E>`

Niebezpieczne

`LinkedList`, `ArrayList`

Biblioteki obiektowe

Sposoby korzystania z klas bibliotecznych

- dziedziczenie z nadklasy;
- implementacja interfejsu.

Dziedziczenie z nadklasy

Przykład 1

```
class Figura
{
    public void narysuj();
    public void przesun(int dx, int dy) { ...}
}
```

Dizedziczenie metody `przesun(int, int)`

Dziedziczenie z nadklasy

Przykład 2

```
public class Thread
{
    public void run();
}
```

Wymóg ponownej implementacji metody `run()`.

Implementacja interfejsu

```
public interface Runnable
{
    void run();
}

public class Thread
{
    public Thread(Runnable target);
}
```

Dziedziczenie z nadklasy

Przykład 3

Implementacja interfejsów graficznych jako podklas klasy `java.awt.Graphics`.

Przykładowe implementacje: aplety, Graphics2D (rysowanie w aplikacjach graficznych).