

Stefan Sokołowski

PROGRAMOWANIE OBIEKTOWE II

wykład 14

**Inst. Informatyki Stosowanej, ANS
Elbląg, 2024/2025**

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie)

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...
- Kiedy wątki
 - *współpracują* przy różnych częściach tego samego zadania, lub
 - *konkurują* o ten sam zasób,powstaje konieczność ich koordynowania dla uniknięcia *wyścigu*.

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...
- Kiedy wątki
 - *współpracują* przy różnych częściach tego samego zadania, lub
 - *konkurują* o ten sam zasób,powstaje konieczność ich koordynowania dla uniknięcia *wyścigu*.
- Sposób Jawy koordynowania wątków *współpracujących*:
 - `t.join()` — czekaj, aż wątek `t` zakończy działanie.

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...
- Kiedy wątki
 - *współpracują* przy różnych częściach tego samego zadania, lub
 - *konkurują* o ten sam zasób,powstaje konieczność ich koordynowania dla uniknięcia *wyścigu*.
- Sposób Jawy koordynowania wątków *współpracujących*:
 - `t.join()` — czekaj, aż wątek `t` zakończy działanie.
- Sposób Jawy koordynowania wątków *konkurujących*:
 - `ob.wait()` — śpij bezczynnie, aż inny wątek cię obudzi

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...
- Kiedy wątki
 - *współpracują* przy różnych częściach tego samego zadania, lub
 - *konkurują* o ten sam zasób,powstaje konieczność ich koordynowania dla uniknięcia *wyścigu*.
- Sposób Jawy koordynowania wątków *współpracujących*:
 - `t.join()` — czekaj, aż wątek `t` zakończy działanie.
- Sposób Jawy koordynowania wątków *konkurujących*:
 - `ob.wait()` — śpij bezczynnie, aż inny wątek cię obudzi;
 - `ob.notify()` — obudź jeden ze śpiących wątków.

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...
- Kiedy wątki
 - *współpracują* przy różnych częściach tego samego zadania, lub
 - *konkurują* o ten sam zasób,powstaje konieczność ich koordynowania dla uniknięcia *wyścigu*.
- Sposób Jawy koordynowania wątków *współpracujących*:
 - `t.join()` — czekaj, aż wątek `t` zakończy działanie.
- Sposób Jawy koordynowania wątków *konkurujących*:
 - `ob.wait()` — śpij bezczynnie, aż inny wątek cię obudzi;
 - `ob.notify()` — obudź jeden ze śpiących wątków.
- Sposób Jawy koordynowania *wszelkich* wątków:
 - `synchronized` — działanie niepodzielne.

Wątki — podsumowanie tego, co dotąd

- Wątki działają *w zasadzie* niezależnie i współbieżnie (jednocześnie);
ale...
- Kiedy wątki
 - *współpracują* przy różnych częściach tego samego zadania, lub
 - *konkurują* o ten sam zasób,powstaje konieczność ich koordynowania dla uniknięcia *wyścigu*.
- Sposób Jawy koordynowania wątków *współpracujących*:
 - `t.join()` — czekaj, aż wątek `t` zakończy działanie.
- Sposób Jawy koordynowania wątków *konkurujących*:
 - `ob.wait()` — śpij bezczynnie, aż inny wątek cię obudzi;
 - `ob.notify()` — obudź jeden ze śpiących wątków.
- Sposób Jawy koordynowania *wszelkich* wątków:
 - `synchronized` — działanie niepodzielne.

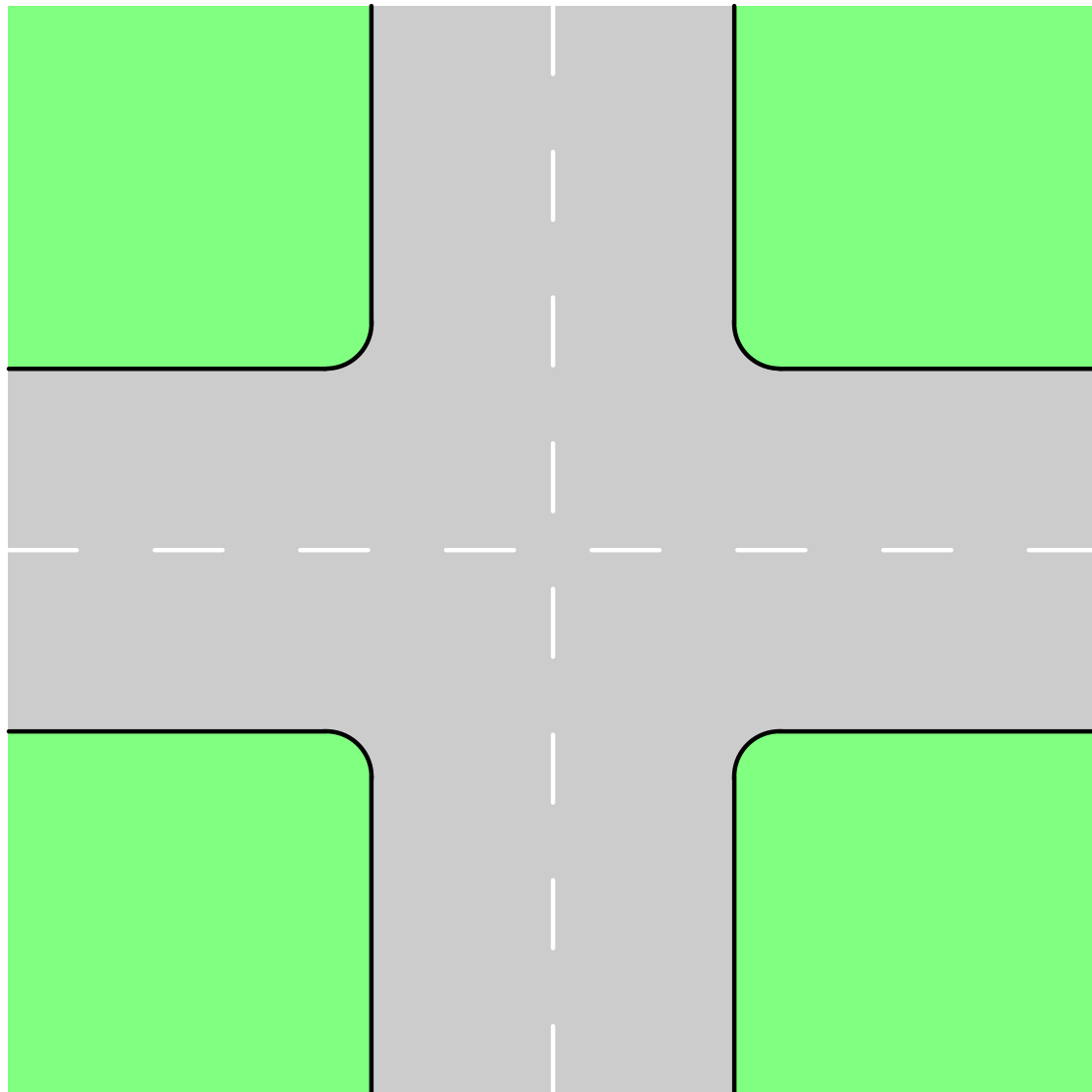
...jednak koordynacja powoduje problemy...

Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

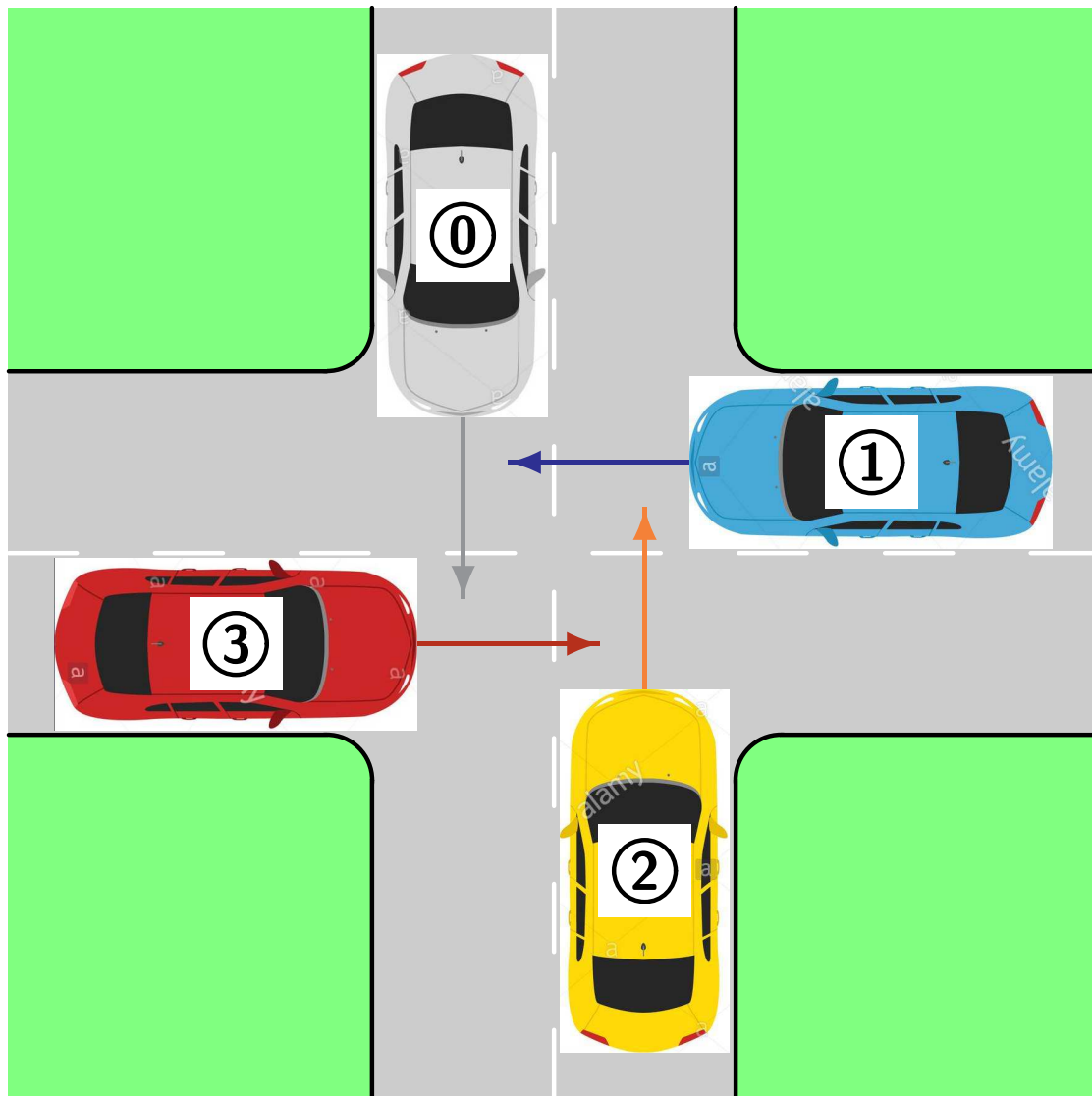
Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*



Niebezpieczeństwa wątkowe

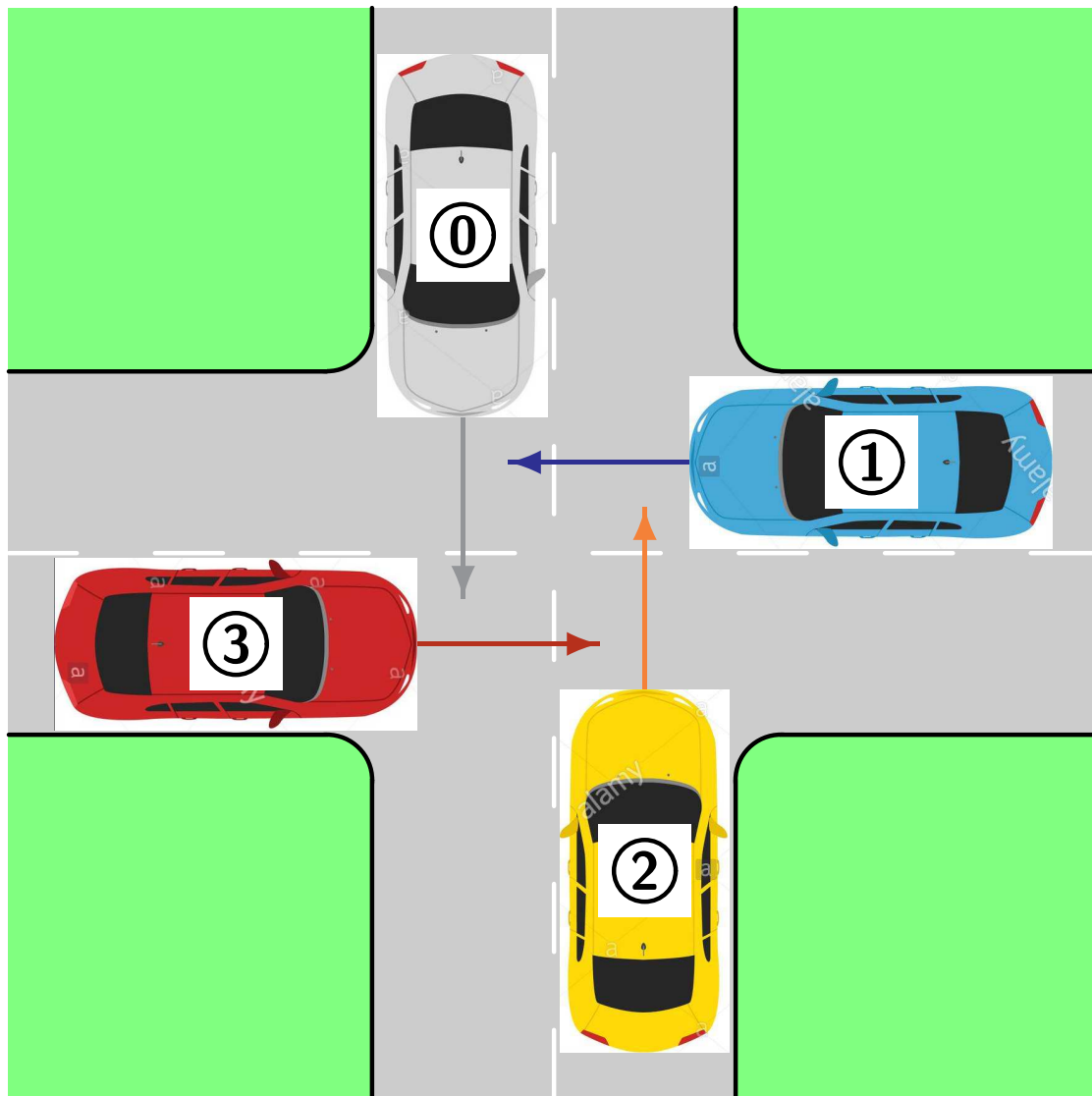
Zakleszczenie — *deadlock*



Wątek (pojazd) blokuje dalsze działanie (przejazd przez skrzyżowanie) innemu wątkowi (pojazdowi z lewej), a sam jest blokowany przez inny wątek (pojazd z prawej).

Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

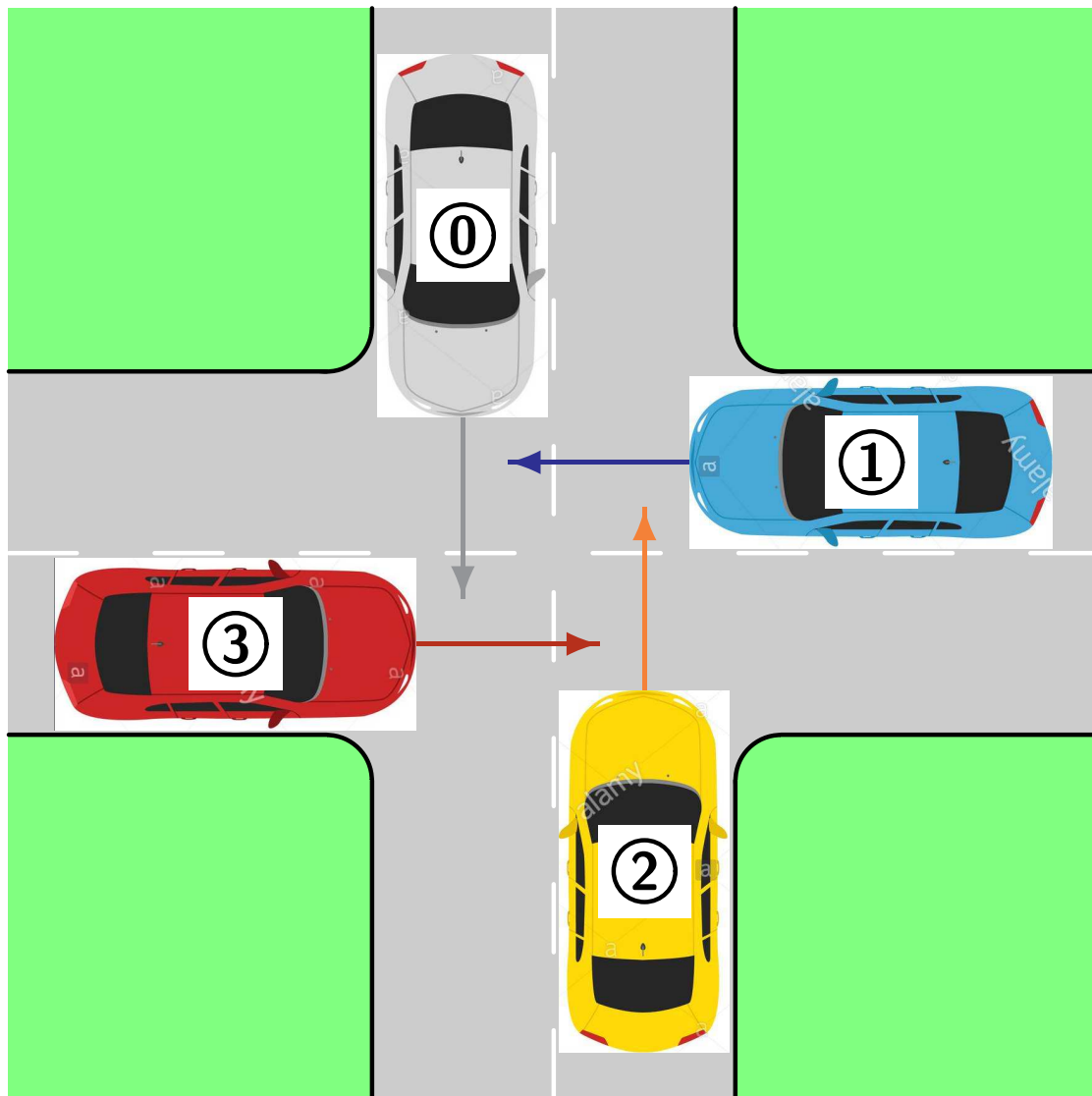


Wątek (pojazd) blokuje dalsze działanie (przejazd przez skrzyżowanie) innemu wątkowi (pojazdowi z lewej), a sam jest blokowany przez inny wątek (pojazd z prawej).

Wszystkie wątki są w stanie **WAITING**; żaden nigdy nie przejdzie w stan **RUNNABLE**.

Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*



Wątek (pojazd) blokuje dalsze działanie (przejazd przez skrzyżowanie) innemu wątkowi (pojazdowi z lewej), a sam jest blokowany przez inny wątek (pojazd z prawej).

Wszystkie wątki są w stanie **WAITING**; żaden nigdy nie przejdzie w stan **RUNNABLE**.

Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

ZAKLESZCZENIE
BLOKADA WZAJEMNA
IMPAS
DEADLOCK

Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

ZAKLESZCZENIE
BLOKADA WZAJEMNA
IMPAS
DEADLOCK

} różne wątki czekają wzajemnie na siebie tak, że żaden nie może działać

Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

ZAKLESZCZENIE
BLOKADA WZAJEMNA
IMPAS
DEADLOCK

różne wątki czekają wzajemnie na siebie tak, że żaden nie może działać

Co to znaczy, że
„komputer się zawiesił” ?



Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

ZAKLESZCZENIE
BLOKADA WZAJEMNA
IMPAS
DEADLOCK

różne wątki czekają wzajemnie na siebie tak, że żaden nie może działać

Co to znaczy, że
„komputer się zawiesił” ?

- albo ślepa pętla (ew. ślepa rekursja)



Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

ZAKLESZCZENIE
BLOKADA WZAJEMNA
IMPAS
DEADLOCK

różne wątki czekają wzajemnie na siebie tak, że żaden nie może działać

Co to znaczy, że
„komputer się zawiesił” ?

- albo ślepa pętla (ew. ślepa rekursja),
- albo właśnie *zakleszczenie*



Niebezpieczeństwa wątkowe

Zakleszczenie — *deadlock*

ZAKLESZCZENIE
BLOKADA WZAJEMNA
IMPAS
DEADLOCK

różne wątki czekają wzajemnie na siebie tak, że żaden nie może działać

Co to znaczy, że
„komputer się zawiesił” ?

- albo ślepa pętla (ew. ślepa rekursja),
- albo właśnie *zakleszczenie*,
- albo inny rodzaj wiecznego czekania.



Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Bank prowadzi wiele kont, których właściciele przelewają sobie wzajemnie pieniądze (w tym programie losowo)

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Bank prowadzi wiele kont, których właściciele przelewają sobie wzajemnie pieniądze (w tym programie losowo):

```
public void run() {  
    while (true) {  
        int id = Bank.los.nextInt(Bank.ileKont),  
            ile = Bank.los.nextInt(Bank.maxPrzekaz);  
        przelewDo(id, ile);  
        try { Thread.sleep(1000); }  
        catch (InterruptedException e) {}  
    }  
}
```

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Bank prowadzi wiele kont, których właściciele przelewają sobie wzajemnie pieniądze (w tym programie losowo):

```
public void run() {  
    while (true) {  
        int id = Bank.los.nextInt(Bank.ileKont),  
            ile = Bank.los.nextInt(Bank.maxPrzekaz);  
        przelewDo(id, ile);  
        try { Thread.sleep(1000); }  
        catch (InterruptedException e) {}  
    }  
}
```



klientowi numer *id*
na kwotę *ile* złotych

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Bank prowadzi wiele kont, których właściciele przelewają sobie wzajemnie pieniądze (w tym programie losowo):

```
public void run() {  
    while (true) {  
        int id = Bank.los.nextInt(Bank.ileKont),  
            ile = Bank.los.nextInt(Bank.maxPrzekaz);  
        przelewDo(id, ile);  
        try { Thread.sleep(1000); }  
        catch (InterruptedException e) {}  
    }  
}
```



klientowi numer **id**
na kwotę **ile** złotych

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Jeśli na koncie nie ma wystarczających środków na przelew, to musi on poczekać, aż nadejdzie jakaś wpłata

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Jeśli na koncie nie ma wystarczających środków na przelew, to musi on poczekać, aż nadejdzie jakaś wpłata:

```
private void przelewDo(int idd, int ile) {
    Boolean czekaj;
    do {
        synchronized(this) {
            if (ile <= stanKonta) {
                wypłata(ile); czekaj = false;
            }
            else czekaj = true;
        }
    } while (czekaj);
    synchronized(Bank.konto[idd]) {
        Bank.konto[idd].wplata(ile);
    }
}
```

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Jeśli na koncie nie ma wystarczających środków na przelew, to musi on poczekać, aż nadejdzie jakaś wpłata:

```
private void przelewDo(int idd, int ile) {  
    Boolean czekaj;  
    do {  
        synchronized(this) {  
            if (ile <= stanKonta) {  
                wypłata(ile); czekaj = false;  
            }  
            else czekaj = true;  
        }  
    } while (czekaj);  
    synchronized(Bank.konto[idd]) {  
        Bank.konto[idd].wplata(ile);  
    }  
}
```

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Jeśli na koncie nie ma wystarczających środków na przelew, to musi on poczekać, aż nadejdzie jakaś wpłata:

```
private void przelewDo(int idd, int ile) {  
    Boolean czekaj;  
    do {  
        synchronized(this) {  
            if (ile <= stanKonta) {  
                wypłata(ile); czekaj = false;  
            }  
            else czekaj = true;  
        }  
    } while (czekaj);  
    synchronized(Bank.konto[idd]) {  
        Bank.konto[idd].wplata(ile);  
    }  
}
```

Może się zdarzyć, że wszyscy klienci czekają, bo nikt nie ma dość pieniędzy, żeby wykonać najbliższy przelew.

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Jeśli na koncie nie ma wystarczających środków na przelew, to musi on poczekać, aż nadejdzie jakaś wpłata:

```
private void przelewDo(int idd, int ile) {  
    Boolean czekaj;  
    do {  
        synchronized(this) {  
            if (ile <= stanKonta) {  
                wypłata(ile); czekaj = false;  
            }  
            else czekaj = true;  
        }  
    } while (czekaj);  
    synchronized(Bank.konto[idd]) {  
        Bank.konto[idd].wplata(ile);  
    }  
}
```

Może się zdarzyć, że wszyscy klienci czekają, bo nikt nie ma dość pieniędzy, żeby wykonać najbliższy przelew. Miałby dosyć, gdyby sam dostał należne przelewy.

Niebezpieczeństwa wątkowe

Zakleszczenie bankowe

Jeśli na koncie nie ma wystarczających środków na przelew, to musi on poczekać, aż nadejdzie jakaś wpłata:

```
private void przelewDo(int idd, int ile) {  
    Boolean czekaj;  
    do {  
        synchronized(this) {  
            if (ile <= stanKonta) {  
                wypłata(ile); czekaj = false;  
            }  
            else czekaj = true;  
        }  
    } while (czekaj);  
    synchronized(Bank.konto[idd]) {  
        Bank.konto[idd].wplata(ile);  
    }  
}
```

Może się zdarzyć, że wszyscy klienci czekają, bo nikt nie ma dość pieniędzy, żeby wykonać najbliższy przelew. Miałby dosyć, gdyby sam dostał należne przelewy.

Takie zakleszczenie gospodarki ekonomicznej nazywają

zatorym płatniczym.

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlocku* nie ma

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlock*u nie ma — chociaż mogą zdarzyć się inne katastrofy.

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlock*u nie ma — chociaż mogą zdarzyć się inne katastrofy.

Deadlock powstaje wtedy, gdy

- dostęp do zasobu jest „ciasny”

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlock*u nie ma — chociaż mogą zdarzyć się inne katastrofy.

Deadlock powstaje wtedy, gdy

- dostęp do zasobu jest „ciasny”, oraz
- nie ma zasad pierwszeństwa (lub są zbyt słabe).

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlocku* nie ma — chociaż mogą zdarzyć się inne katastrofy.

Deadlock powstaje wtedy, gdy

- dostęp do zasobu jest „ciasny”, oraz
- nie ma zasad pierwszeństwa (lub są zbyt słabe).

Przykład: (skrzyżowanie)

Gdyby

- któryś z pojazdów był uprzywilejowany, albo
- pojazdy z jednej drogi miały pierwszeństwo,

to do *deadlocku* by nie doszło.

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlocku* nie ma — chociaż mogą zdarzyć się inne katastrofy.

Deadlock powstaje wtedy, gdy

- dostęp do zasobu jest „ciasny”, oraz
- nie ma zasad pierwszeństwa (lub są zbyt słabe).

Przykład: (skrzyżowanie)

Gdyby

- któryś z pojazdów był uprzywilejowany, albo
- pojazdy z jednej drogi miały pierwszeństwo,

to do *deadlocku* by nie doszło.

Przykład: (bank)

Zatory płatnicze zwykle rozwiązuje rząd przez „dodruk” pieniędzy.

Niebezpieczeństwa wątkowe

Niezależnie działające wątki konkurują o wspólny zasób.

Jeśli nie ma między nimi **żadnej** koordynacji ograniczającej dostęp do zasobu, to *deadlocku* nie ma — chociaż mogą zdarzyć się inne katastrofy.

Deadlock powstaje wtedy, gdy

- dostęp do zasobu jest „ciasny”, oraz
- nie ma zasad pierwszeństwa (lub są zbyt słabe).

Przykład: (skrzyżowanie)

Gdyby

- któryś z pojazdów był uprzywilejowany, albo
- pojazdy z jednej drogi miały pierwszeństwo,

to do *deadlocku* by nie doszło.

Przykład: (bank)

Zatory płatnicze zwykle rozwiązuje rząd przez „dodruk” pieniędzy.

Niebezpieczeństwa wątkowe

Wieczne czekanie, ale nie *deadlock*

```
class Watek implements Runnable {
    public void run() {
        synchronized(this) {
            try { this.wait(); }
            catch (InterruptedException e) {}
        }
    }
}

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

Niebezpieczeństwa wątkowe

Wieczne czekanie, ale nie *deadlock*

```
class Watek implements Runnable {
    public void run() {
        synchronized(this) {
            try { this.wait(); }
            catch(InterruptedException e){}
        }
    }
}

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

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog14/A4-InneWiszenie.java

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy

zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy
zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Więc w ciągu doby lekarz poświęca:

na zawały	
na udary	
na złamania	
razem	0 ^h

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Więc w ciągu doby lekarz poświęca:

na zawały	
na udary	
na złamania	
razem	0 ^h

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Więc w ciągu doby lekarz poświęca:

na zawały	16 ^h
na udary	
na złamania	
razem	16 ^h

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Więc w ciągu doby lekarz poświęca:

na zawały	16 ^h
na udary	6 ^h
na złamania	
razem	22 ^h

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Więc w ciągu doby lekarz poświęca:

na zawały	16 ^h
na udary	6 ^h
na złamania	2 ^h
razem	24^h

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Szpitalny Oddział Ratunkowy zajmuje się takimi przypadkami:

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

W ciągu doby zdarza się średnio:

	liczba przypadków
zawałów	2
udarów	3
złamań	4

Więc w ciągu doby lekarz poświęca:

na zawały	16 ^h
na udary	6 ^h
na złamania	2 ^h
razem	24^h

Zawały mają pierwszeństwo przed udarami a udary przed złamaniami.

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa chory:

```
class Chory implements Runnable {  
    . . . . .  
    public void run() {  
        System.out.print(choroba.nazwa() + " " + id + " nowy]" );  
        Main.doKolejki(this);  
        try { synchronized(this) {this.wait(); } }  
        catch(InterruptedException e) { };  
        System.out.print(choroba.nazwa() + " " + id + " juz" );  
    }  
}
```

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa chory:

```
class Chory implements Runnable {  
    . . . . .  
    public void run() {  
        System.out.print(choroba.nazwa() + " " + id + " nowy]" );  
        Main.doKolejki(this);  
        try { synchronized(this) {this.wait(); } }  
        catch(InterruptedException e) { };  
        System.out.print(choroba.nazwa() + " " + id + " juz" );  
    }  
}
```

do kolejki
i czekać

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa lekarz:

```
class Lekarz implements Runnable {  
    public void run() {  
        while (true) {  
            Chory ch = Main.zKolejki();  
            try {  
                Thread.sleep((int)ch.choroba().czasLekarza());  
            }  
            catch (InterruptedException e) {};  
            synchronized(ch) { ch.notify(); }  
        }  
    }  
}
```

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa lekarz:

```
class Lekarz implements Runnable {  
    public void run() {  
        while (true) {  
            Chory ch = Main.zKolejki();  
            try {  
                Thread.sleep((int)ch.choroba().czasLekarza());  
            }  
            catch (InterruptedException e) {}  
            synchronized(ch) { ch.notify(); }  
        }  
    }  
}
```

pobrać pacjenta
z kolejki



Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa lekarz:

```
class Lekarz implements Runnable {  
    public void run() {  
        while (true) {  
            Chory ch = Main.zKolejki();  
            try {  
                Thread.sleep((int)ch.choroba().czasLekarza());  
            }  
            catch (InterruptedException e) {};  
            synchronized(ch) { ch.notify(); }  
        }  
    }  
}
```

pobrać pacjenta
z kolejki

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa lekarz:

```
class Lekarz implements Runnable {  
    public void run() {  
        while (true) {  
            Chory ch = Main.zKolejki();  
            try {  
                Thread.sleep((int)ch.choroba().czasLekarza());  
            }  
            catch (InterruptedException e) {}  
            synchronized(ch) { ch.notify(); }  
        }  
    }  
}
```

pobrać pacjenta z kolejki

puścić pacjenta do domu

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak działa lekarz:

```
class Lekarz implements Runnable {  
    public void run() {  
        while (true) {  
            Chory ch = Main.zKolejki();  
            try {  
                Thread.sleep((int)ch.choroba().czasLekarza());  
            }  
            catch (InterruptedException e) {}  
            synchronized(ch) { ch.notify(); }  
        }  
    }  
}
```

pobrać pacjenta z kolejki

puścić pacjenta do domu

iis.ans-elblag.pl/~stefan/Dydaktyka/ProgObiII/Prog14/W1_S0R.java

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak zaplanowano: 

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Tak zaplanowano: 

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h
złamanie	0 ^h 30'

[zawał 0 nowy] [udar 2 nowy] [złamanie 1 nowy] [złamanie 3 nowy] [udar 4 nowy]
 zawał 0 juz
 udar 2 juz [zawał 5 nowy] [złamanie 6 nowy]
 udar 4 juz [udar 7 nowy] [złamanie 8 nowy]
 zawał 5 juz
 udar 7 juz
 złamanie 1 juz [zawał 9 nowy] [udar 10 nowy]
 złamanie 3 juz [złamanie 11 nowy] [złamanie 12 nowy]
 zawał 9 juz [udar 13 nowy]
 udar 10 juz [zawał 14 nowy]
 udar 13 juz [złamanie 15 nowy] [udar 16 nowy] [złamanie 17 nowy]
 zawał 14 juz

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

a tak wyszło, bo udary
potrzebowały więcej czasu →

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h 40'
złamanie	0 ^h 30'

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

a tak wyszło, bo udary
potrzebowały więcej czasu →

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h 40'
złamanie	0 ^h 30'

[udar 2 nowy] [złamanie 1 nowy] [zawał 0 nowy]
 udar 2 juz [złamanie 3 nowy] [udar 4 nowy]
 zawał 0 juz [zawał 5 nowy] [złamanie 6 nowy]
 udar 4 juz [udar 7 nowy] [złamanie 8 nowy]
 zawał 5 juz [zawał 9 nowy] [udar 10 nowy] [złamanie 11 nowy]
 udar 7 juz [złamanie 12 nowy] [udar 13 nowy]
 zawał 9 juz
 udar 10 juz [zawał 14 nowy] [złamanie 15 nowy]
 udar 13 juz [udar 16 nowy] [złamanie 17 nowy]
 zawał 14 juz [zawał 18 nowy] [udar 19 nowy] [złamanie 20 nowy]
 udar 16 juz [złamanie 21 nowy] [udar 22 nowy]
 zawał 18 juz

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

a tak wyszło, bo udary
potrzebowały więcej czasu →

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h 40'
złamanie	0 ^h 30'

[udar 2 nowy] [złamanie 1 nowy] [zawał 0 nowy]
 udar 2 juz [złamanie 3 nowy] [udar 4 nowy]
 zawał 0 juz [zawał 5 nowy] [złamanie 6 nowy]
 udar 4 juz [udar 7 nowy] [złamanie 8 nowy]
 zawał 5 juz [zawał 9 nowy] [udar 10 nowy] [złamanie 11 nowy]
 udar 7 juz [złamanie 12 nowy] [udar 13 nowy]
 zawał 9 juz
 udar 10 juz [zawał 14 nowy] [złamanie 15 nowy]
 udar 13 juz [udar 16 nowy] [złamanie 17 nowy]
 zawał 14 juz [zawał 18 nowy] [udar 19 nowy] [złamanie 20 nowy]
 udar 16 juz [złamanie 21 nowy] [udar 22 nowy]
 zawał 18 juz

Złamania nadal się zdarzają, ale już nie są leczone: są zawsze wyprzedzane przez zawały i udary.

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

a tak wyszło, bo udary
potrzebowały więcej czasu →

	czas pracy lekarza
zawał	8 ^h
udar	2 ^h 40'
złamanie	0 ^h 30'

[udar 2 nowy] [złamanie 1 nowy] [zawał 0 nowy]
 udar 2 juz [złamanie 3 nowy] [udar 4 nowy]
 zawał 0 juz [zawał 5 nowy] [złamanie 6 nowy]
 udar 4 juz [udar 7 nowy] [złamanie 8 nowy]
 zawał 5 juz [zawał 9 nowy] [udar 10 nowy] [złamanie 11 nowy]
 udar 7 juz [złamanie 12 nowy] [udar 13 nowy]
 zawał 9 juz
 udar 10 juz [zawał 14 nowy] [złamanie 15 nowy]
 udar 13 juz [udar 16 nowy] [złamanie 17 nowy]
 zawał 14 juz [zawał 18 nowy] [udar 19 nowy] [złamanie 20 nowy]
 udar 16 juz [złamanie 21 nowy] [udar 22 nowy]
 zawał 18 juz

Złamania nadal się zdarzają, ale już nie są leczone: są zawsze wyprzedzane przez zawały i udary. Wątki pacjentów ze złamaniami nigdy nie zostaną obudzone.

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Z **zagłodzeniem** wątku mamy do czynienia wtedy, gdy w konkurencji do zasobu (tutaj: do lekarza) jest bez końca wyprzedzany przez inne wątki i nigdy nie zostaje do niego dopuszczony.

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Z **zagłodzeniem** wątku mamy do czynienia wtedy, gdy w konkurencji do zasobu (tutaj: do lekarza) jest bez końca wyprzedzany przez inne wątki i nigdy nie zostaje do niego dopuszczony.

Zagłodzenie bierze się na ogół z ustawienia priorytetów dostępu do zasobu.

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Z **zagłodzeniem** wątku mamy do czynienia wtedy, gdy w konkurencji do zasobu (tutaj: do lekarza) jest bez końca wyprzedzany przez inne wątki i nigdy nie zostaje do niego dopuszczony.

Zagłodzenie bierze się na ogół z ustawienia priorytetów dostępu do zasobu.

Na przykład może być tak:

— wątek o niskim priorytecie jest zawsze wyprzedzany przez wątki o wyższym priorytecie (tutaj: złamania wyprzedzane przez zawały i udary).

Niebezpieczeństwa wątkowe

Zagłodzenie — *starvation*

Z **zagłodzeniem** wątku mamy do czynienia wtedy, gdy w konkurencji do zasobu (tutaj: do lekarza) jest bez końca wyprzedzany przez inne wątki i nigdy nie zostaje do niego dopuszczony.

Zagłodzenie bierze się na ogół z ustawienia priorytetów dostępu do zasobu.

Na przykład może być tak:

— wątek o niskim priorytecie jest zawsze wyprzedzany przez wątki o wyższym priorytecie (tutaj: złamania wyprzedzane przez zawały i udary).

Albo tak:

— zawsze dajemy pierwszeństwo wątkowi mniejszemu, który zajmie zasób na krócej.

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie.

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu*

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu* (np. drukarka drukuje dwa wymieszane teksty na raz)

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu* (np. drukarka drukuje dwa wymieszane teksty na raz);
- **są** skoordynowane, ale bez dobrze ustawionych praw pierwszeństwa, to może dojść do *zakleszczenia*

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu* (np. drukarka drukuje dwa wymieszane teksty na raz);
- **są** skoordynowane, ale bez dobrze ustawionych praw pierwszeństwa, to może dojść do *zakleszczenia* (np. system wzajemnych płatności zamiera)

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu* (np. drukarka drukuje dwa wymieszane teksty na raz);
- **są** skoordynowane, ale bez dobrze ustawionych praw pierwszeństwa, to może dojść do *zakleszczenia* (np. system wzajemnych płatności zamiera);
- **są** skoordynowane, z prawami pierwszeństwa zbyt silnie faworyzującymi niektóre wątki, to może dojść do *zagłodzenia*

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu* (np. drukarka drukuje dwa wymieszane teksty na raz);
- **są** skoordynowane, ale bez dobrze ustawionych praw pierwszeństwa, to może dojść do *zakleszczenia* (np. system wzajemnych płatności zamiera);
- **są** skoordynowane, z prawami pierwszeństwa zbyt silnie faworyzującymi niektóre wątki, to może dojść do *zagłodzenia* (np. system działa, ale niektóre śpiące wątki nigdy nie są budzone).

Niebezpieczeństwa wątkowe

W zasadzie programista, tworząc wątek, myśli o nim jak o samodzielnym niezależnym programie. Ale jeśli wspólnie działające wątki

- **nie są** skoordynowane, to może dojść do *wyścigu* (np. drukarka drukuje dwa wymieszane teksty na raz);
- **są** skoordynowane, ale bez dobrze ustawionych praw pierwszeństwa, to może dojść do *zakleszczenia* (np. system wzajemnych płatności zamiera);
- **są** skoordynowane, z prawami pierwszeństwa zbyt silnie faworyzującymi niektóre wątki, to może dojść do *zagłodzenia* (np. system działa, ale niektóre śpiące wątki nigdy nie są budzone).

Programowanie współbieżne jest trudne. Łatwo wpaść z jednego niebezpieczeństwa w inne.