Nowoczesne języki programowania obiektowego

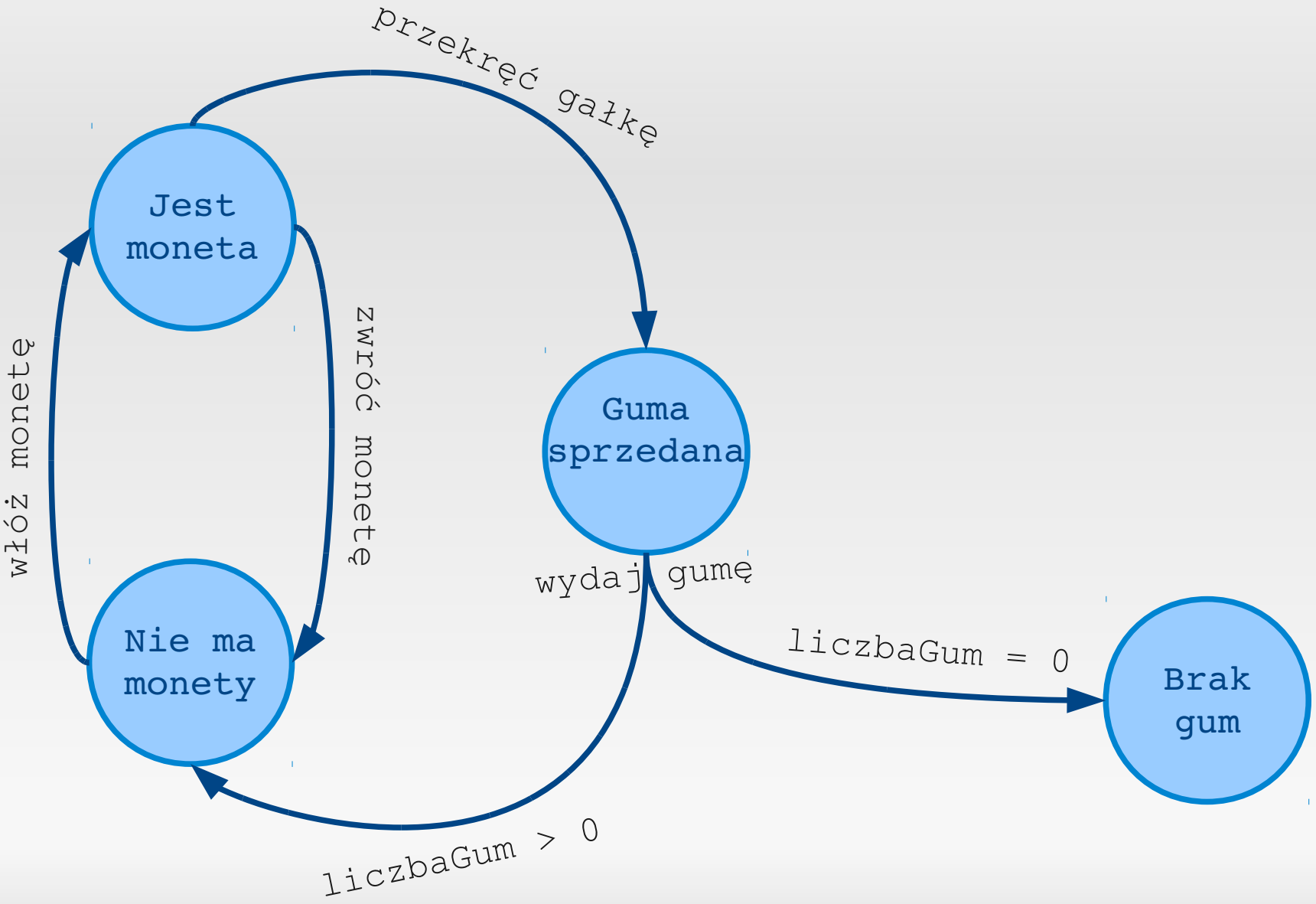
Wzorce projektowe: STAN

dr Aleksander Lamża

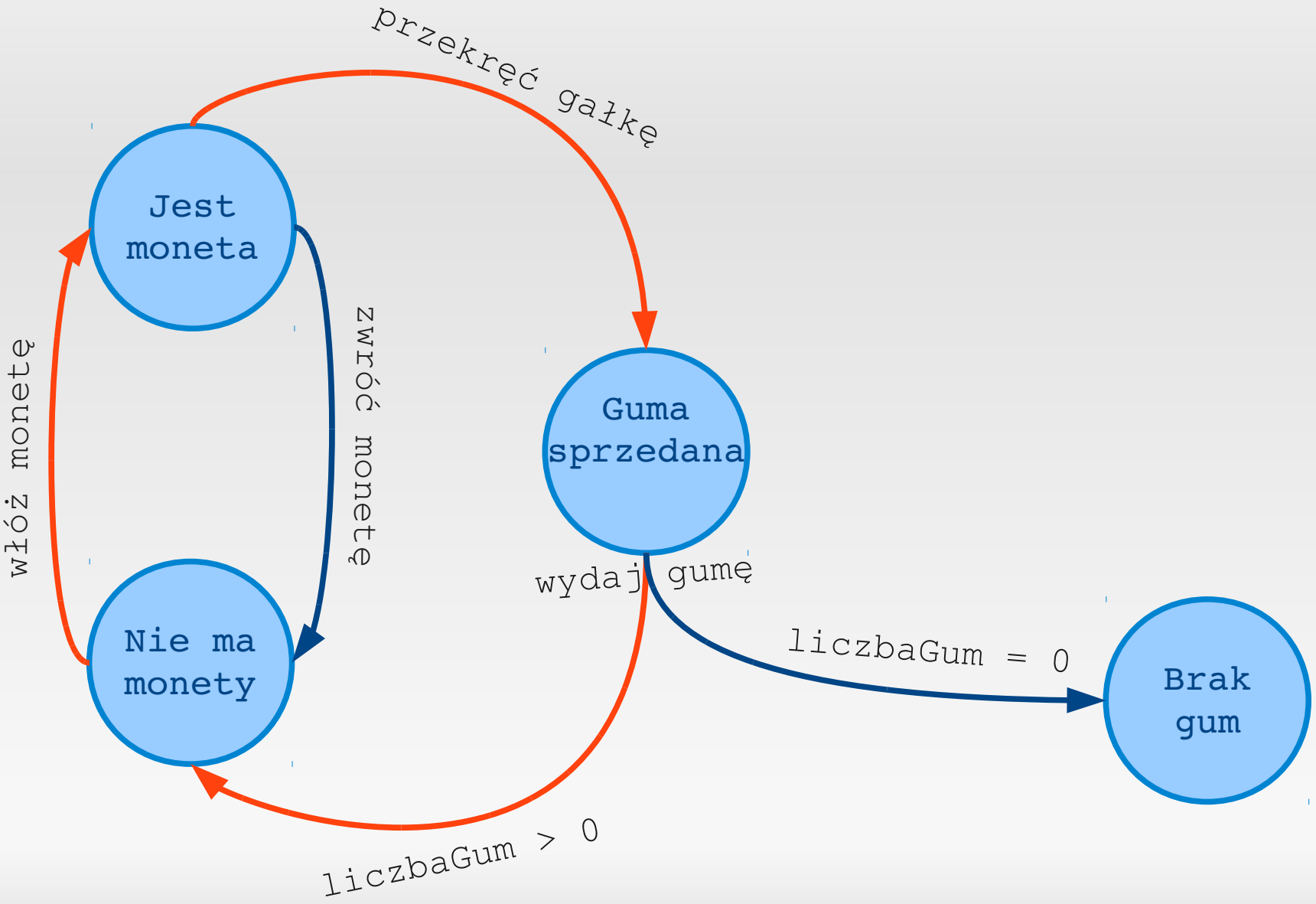
Wydział Informatyki i Nauki o Materiałach
Instytut Informatyki, Uniwersytet Śląski

Misja: oprogramować automat do sprzedaży gum do żucia

Jak to działa? (diagram stanów)



Jak to działa? (typowe działanie)



Implementacja - stany

STANY:


```
final static int BRAK_GUM = 0;
final static int NIE_MA_MONETY = 1;
final static int JEST_MONETA = 2;
final static int GUMA_SPRZEDANA = 3;

int stan = BRAK_GUM;
```

Implementacja - operacje

OPERACJE:

włóż monetę

zwróć monetę

przekręć gałkę

wydaj gumę

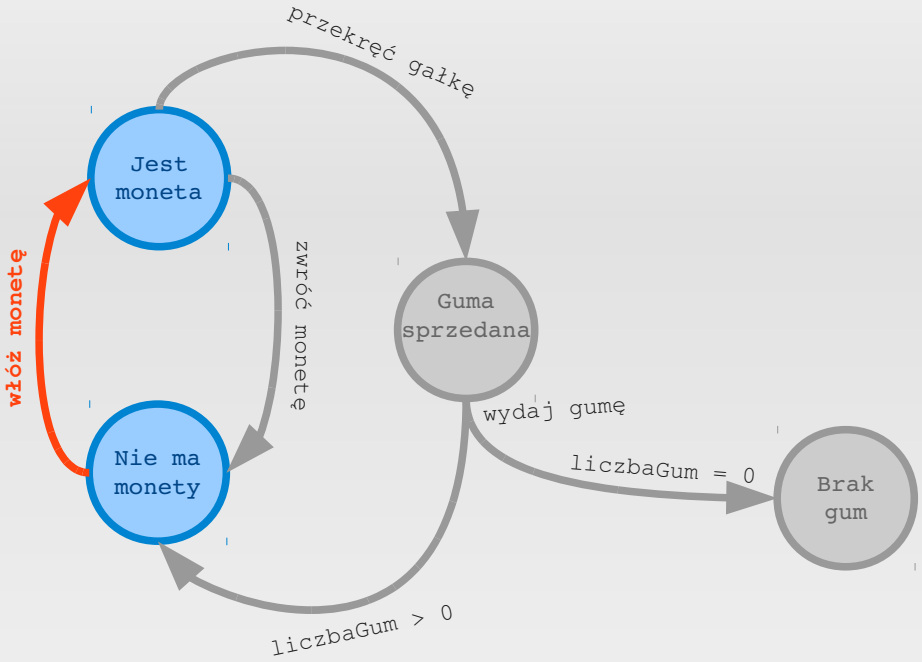
```
public void wlozMonete() {
   ...
}

public void zwrocMonete() {
   ...
}

public void przekrecGalke() {
   ...
}

public void wydajGume() {
   ...
}
```

Implementacja pojedynczej operacji




```
public void wlozMonete() {
  if (stan == JEST_MONETA) {
      println("Po co druga moneta?");
  } else if (stan == NIE_MA_MONETY) {
      println("Moneta przyjęta");
      stan = JEST_MONETA;
  } else if (stan == BRAK_GUM) {
      println("Automat jest pusty!");
  } else if (stan == GUMA_SPRZEDANA) {
      println("Czekaj na gumę!");
  }
}
```

Klasa AutomatSprzedajacy

```
public class AutomatSprzedajacy {

  final static int BRAK_GUM = 0;
  final static int NIE_MA_MONETY = 1;
  final static int JEST_MONETA = 2;
  final static int GUMA_SPRZEDANA = 3;

  int stan = BRAK_GUM;
  int liczbaGum = 0;

  public void wlozMonete() {
    ...
  }

  public void zwrocMonete() {
    ...
  }

  public void przekrecGalke() {
    ...
  }

  public void wydajGume() {
    ...
  }
}
```

STANY

OPERACJE

Uzupełnienie klasy + main()

- Konstruktor **`AutomatSprzedajacy(int liczbaGum)`**.
- Metoda **`toString()`** zwracająca String ze stanem automatu i liczbą gum.

```
public static void main(String[] args) {
  AutomatSprzedajacy automat = new AutomatSprzedajacy(3);

  System.out.println(automat);  // NIE_MA_MONETY 3

  // typowe działanie
  automat.wlozMonete();
  automat.przekrecGalke();

  System.out.println(automat);  // GUMA_SPRZEDANA 2

  // uwaga, oszust!
  automat.wlozMonete();
  automat.zwrocMonete();
  automat.przekrecGalke();

  System.out.println(automat);  // NIE_MA_MONETY 2

  ...
}
```

Implementacja całości

ciąg dalszy nastąpi...

Kilka kłopotliwych pytań

- Czy ten kod spełnia zasadę *otwarte-zamknięte* (otwarty na rozbudowę, ale zamknięty na modyfikacje)?
- Czy w tym projekcie zostały wykorzystane możliwości, jakie daje OOP?
- Czy utrzymanie takiego kodu jest proste?
- Czy kod jest czytelny i przejrzysty?
- Co trzeba będzie zrobić, by wprowadzić nowy stan?
- itd.

Czas na ZMIANY!

- Skoro aplikacja jest już gotowa, przetestowana i wdrożona, czas na **ZMIANY**.
- Zleceniodawca postanowił wprowadzić ”drobną” modyfikację...

- Jak to zrealizować?

Na początek wielka refaktoryzacja

- Każdy stan w osobnej **klasie**.
- Klasa abstrakcyjna lub interfejs jako baza dla klas stanów.
- W klasie **`AutomatSprzedajacy`** zamiast stałych powinna znajdować się **zmienna** (zmienne) dla **obiektów stanów**.

Wzorzec STAN umożliwia obiektowi zmianę zachowania wraz ze zmianą jego wewnętrznego stanu. Po takiej zmianie funkcjonuje on jak obiekt innej klasy.

Kolejny stan - WYGRANA