Използване на цикли и други команди

Правилна организация на реда на изпълнението

Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

Съдържание

- Организация на праволинеен код
- Използване на цикли
- Други структури за реда на изпълнение

Праволинеен код

- Когато редът на командите е важен

```
GetData();
GroupData();
Print();
```

  - Направете зависимостите очевидни
  - Именувайте методите според тези зависимости
  - Използвайте параметри на методи

```
data = GetData();
groupedData = GroupData(data);
PrintGroupedData(groupedData);
```

  - Ако е необходимо документирайте реда на изпълнение

Праволинеен код (2)

- Когато редът на командите не е важен
  - Нека четенето на кода отгоре до долу да е като четенето на вестник
  - Групирайте свързани команди заедно
  - Поставете ясни граници при зависими команди
    - Ползвайте празни редове за да отделите зависимостите
    - Ползвайте отделен метод

Праволинеен код – примери

```
ReportFooter CreateReportFooter(Report report)
{
    // …
}

ReportHeader CreateReportHeader(Report report)
{
    // …
}

Report CreateReport()
{
    var report = new Report();
    report.Footer = CreateReportFooter(report);
    report.Content = CreateReportContent(report);

    report.Header = CraeteReportHeader(report);
    return report;
}

ReportContent CreateReportContent(Report report)
{
    // …
}
```

Праволинеен код – примери (2)

```
Report CreateReport()
{
  var report = new Report();

  report.Header = CreateReportHeader(report);
  report.Content = CreateReportContent(report);
  report.Footer = CreateReportFooter(report);

  return report;
}

ReportHeader CreateReportHeader(Report report)
{
  // …
}

ReportContent CreateReportContent(Report report)
{
  // …
}

ReportFooter CreateReportFooter(Report report)
{
  // …
}
```

Използване на цикли

- Избирането на правилния тип цикъл:
  - Ползвайте цикъла **`for`** да повторите блок от код определен брой пъти
  - Ползвайте цикъла **`foreach`** да извършите действие с всеки елемент от масив или колекция
  - Ползвайте циклите **`while`** / **`do-while`** когато не знаете колко пъти трябва да повторите блока команди
- Избягвайте многократното влагане на цикли
  - Можете да изнесете тялото на цикъла в нов метод

Цикли: Най-добри практики

- Нека циклите са прости
  - Това подпомага тези, които ще четат вашия код
- Отнасяйте се с вътрешността на цикъла като с процедура
  - Не карайте четящия да гледа в цикъла, за да разбере управлението му
- Отнасяйте се към цикъла като към „черна кутия“:

```
while (!inputFile.EndOfFile() && !hasErrors)
{
        (black box code)
}
```

Цикли: Най-добри практики (2)

- Актуализацията е добре да е в началото или края на блока команди

```
while (index < 10)
{
    ...
    index += 2;
    ...
}
```

```
while (index < 10)
{
    ...
    ...
    index += 2;
}
```

- Ползвайте смислени имена за променливите, за да направите цикъла лесен за разчитане

```
for(i = 2000; i < 2011; i++)
{
    for(j = 1; j <= 12; j++)
    ...
}
```

```
for (year = 2000; year < 2011; year++)
{
    for (month = 1; month <= 12; month++)
    ...
}
```

Цикли: Най-добри практики (3)

- Избягвайте празни цикли

```
while ((inputChar = Console.Read()) != '\n')
{
}
```

```
do
{
    inputChar = Console.Read();
}
while (inputChar != '\n');
```

- Вземете в предвид семантиките за циклите в съответния език
  - C# – внимавайте за access to modified closure

Цикли: Съвети за цикъл for

- Не променяйте нарочно стойността на индекса за да принудите цикъла да спре
  - Вместо това ползвайте цикъл **while** с команда **break**
- В заглавната част на цикъла слагайте само изразите, които го контролират

```
for (i = 0, sum = 0;
  i < length;
  sum += arr[i], i++)
{
    ;
}
```

```
sum = 0;
for (i = 0; i < length; i++)
{
    sum += arr[i];
}
```

Цикли: Съвети за цикъл for (2)

- Избягвайте код, зависим от крайната стойност на индекса

```
for (i = 0; i < length; i++)
{
  if (array[i].Id == key)
  {
    break;
  }
}

// Lots of code
...

return (i < length);
```

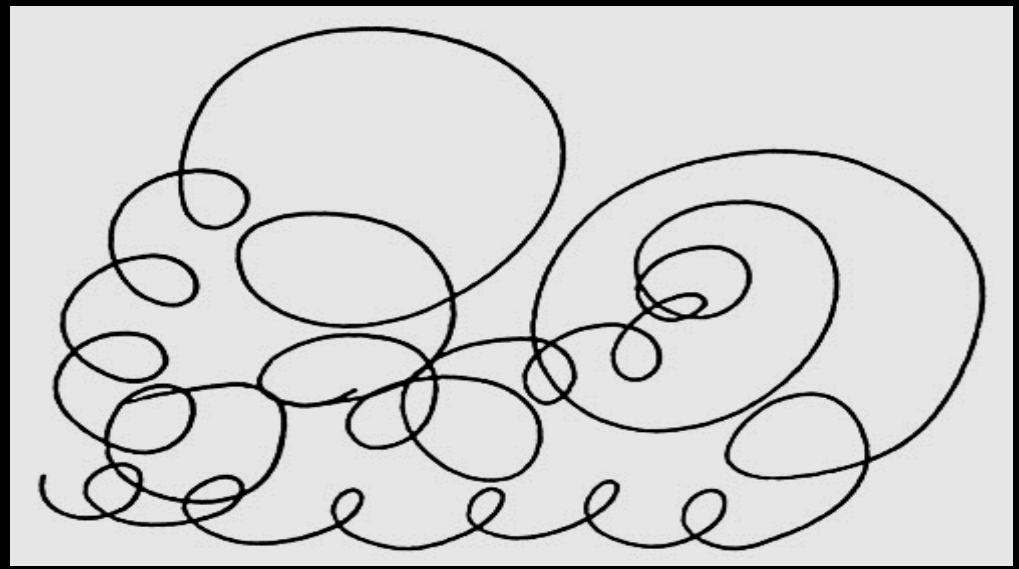
```
bool found = false;
for (i = 0; i < length; i++)
{
  if (array[i].Id == key)
  {
    found = true;
    break;
  }
}

// Lots of code
...

return found;
```

Цикли: Break и Continue

- Ползвайте `continue` за тестове в началото на цикъл, за да избегнете вмъкнати `if`-ове
- Избягвайте цикли с много команди `break` в тях
- Бъдете внимателни когато ползвате `break` и `continue`

Колко дълъг трябва да е един цикъл?

- Опитвайте да направите циклите толкова кратки, че да се виждат наведнъж (на един екран)
- Ползвайте методи да съкратите тялото на цикъла
- Правете дългите цикли особено ясни
- Избягвайте многократното влагане на цикли



Изразът Return

- Ползвайте **return** когато това би подобрило четливостта
- Ползвайте **return** за да избегнете многократното влагане

```
void ParseString(string str)
{
    if (string != null)
    {
        // Lots of code
    }
}
```

```
void ParseString(string str)
{
    if (string == null)
    {
        return;
    }

    // Lots of code
}
```

- Избягвайте честата употреба на **return** в дълги методи

GOTO

- Избягвайте командата **`goto`**, защото тя може да доведе до „спагети код“
- “A Case Against the GO TO Statement” от Edsger Dijkstra

- Ползвайте **`goto`** само в краен случай
  - Ако прави кода по-лесен за поддръжка
- C# поддържа **`goto`** с етикети, но го избягвайте!

Обобщение

1. Праволинеен код
   - Подреждайте методите според тяхната зависимост
2. Команди за цикли
   - Дръжте изразите и командите прости
   - Ползвайте подходящи структури за контрол на реда на действията
   - Не ползвайте **`goto`**

Използване на цикли и други команди

Въпроси?

https://it-kariera.mon.bg/e-learning/

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**