Отражение на типове (Reflection)

Учителски екип

Обучение за ИТ кариера

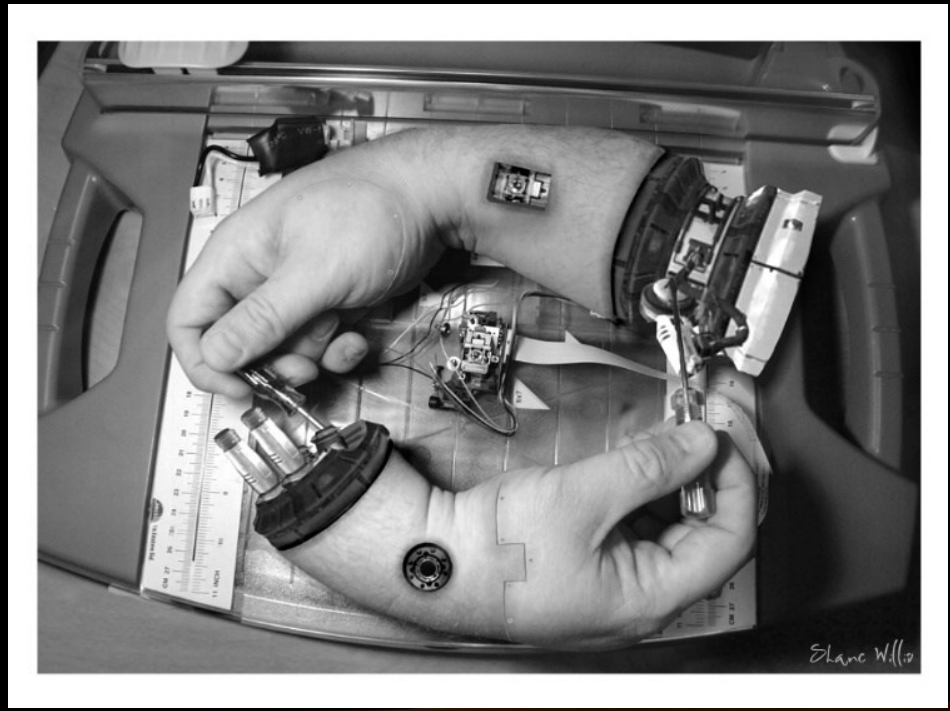
https://it-kariera.mon.bg/e-learning/

Съдържание

1. Какво? Защо? Къде? Кога?
2. Reflection API
   - Клас Type
   - Отражение на полета
   - Отражение на конструктори
   - Отражение на методи

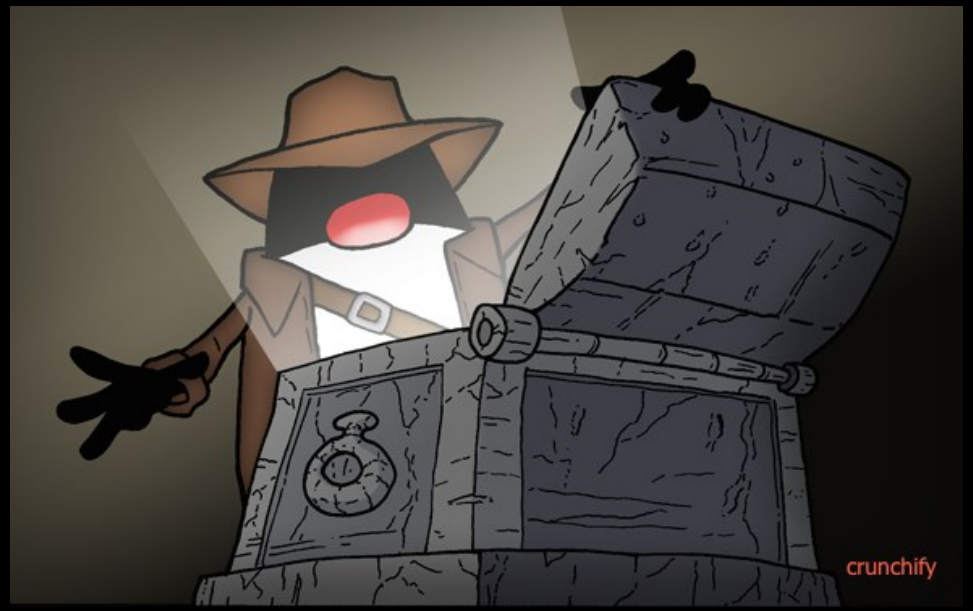
Какво е Метапрограмиране?

- Техника на програмиране, при която компютърни програми мога да третират други програми като свои данни
- Програмите може да са проектирани да:
  - Четат
  - Генерират
  - Анализират
  - Трансформират
- Променяйки се в движение



Какво е "отражение на типовете" (Reflection)?

- Способността на програмен език да бъде свой собствен метаезик
- Програмите може да проверява информация за себе си



Защо и къде да ползваме отражението?

- Кодът става по-разширяем
- Намалява значително дължината на кода
- По-лесна поддръжка
- Тестване
- Инструменти за програмисти

Кога да се ползва отражението?

- Ако е възможно да се извърши дадена операция без използването на отражение, се препоръчва то да се избягва
- Негативи на отражението
  - Бързодействието страда
  - Ограничения в сигурността
  - Излагане на вътрешната структура

Клас Type

- Основен начин за достъп до **метаданните**
- Извлича се по време на компилиране, ако е ясно името

```
Type myType = typeof(ClassName);
```

- Извлича се по време на изпълнение, ако името е неизвестно

```
Type myType = Type.GetType("Namespace.ClassName")
```

Трябва ви напълно определено (fully qualified) име на класа като низ

Клас Name

- Предоставя името на **класа**
  - Напълно определеното име - **Type.FullName**

```
string fullName = typeOf(SomeClass).FullName;
```

  - Името на класа без името на пакета - **Type.Name**

```
string simpleName = typeOf(SomeClass).Name;
```

Основен клас и интерфейси

- Предоставя основния клас

```
Type baseType = testClass.BaseType;
```

- Предоставя интерфейсите

```
Type[] interfaces = testClass.GetInterfaces();
```

  - Връща само интерфейсите, изрично декларирани като имплементирани от дадения клас

Нова инстанция

- **CreateInstance** – създава екземпляр от класа чрез извикване на конструктора, който най-добре пасва на указаните параметри

```
Type sbType =
    Type.GetType("System.Text.StringBuilder");

StringBuilder sbInstance =
    (StringBuilder) Activator.CreateInstance(sbType);
StringBuilder sbInstCapacity = (StringBuilder)Activator
   .CreateInstance(sbType, new object[] {10});
```

Отражение на полета

- Предоставя публичните полета

```
FieldInfo field = type.GetField("name");
FieldInfo[] publicFields = type.GetFields();
```

- Предоставя всички полета

```
FieldInfo[] allFields = type.GetFields(
BindingFlags.Static | BindingFlags.Instance |
BindingFlags.Public | BindingFlags.NonPublic);
```

Тип и име на полето

- Получаване на името и типа на полето

```
FieldInfo field = type.GetField("fieldName");
string fieldName = field.Name;
Type fieldType = field.FieldType;
```

Промяна на поле

```
Type testType = typeof(Test);
Test testInstance =
    (Test) Activator.CreateInstance(testType);
FieldInfo field = testType.GetField("testInt");

field.SetValue(testInstance, 5);
int fieldValue =
     (int) field.GetValue(testInstance);
```

Внимание: използвайте много внимателно, тъй като може да промените вътрешното състояние на обекта!

Модификатори на достъпа

- Всеки модификатор е флаг (1 бит) който е вярно / невярно
- Проверка на модификаторите за достъп на член на класа

```
field.IsPrivate      //частен
field.IsPublic       //публичен
field.IsNonPublic    //не е публичен
field.IsFamily       //защитен (protected)
field.IsAssembly     //вътрешен (internal)
и т.н.…
```

Отражение на конструкторите

- Предоставя конструкторите

```
ConstructorInfo[] publicCtors =
                              type.GetConstructors();
```

- Предоставя всички нестатични конструктори

```
ConstructorInfo[] allNonStaticCtors =
    type.GetConstructors(
                  BindingFlags.Instance |
                  BindingFlags.Public |
                  BindingFlags.NonPublic);
```

Отражение на конструкторите (2)

- Достъп до отделен конструктор

```
ConstructorInfo constructor =
    type.GetConstructor(new Type[] parametersType);
```

- Получаване на типа на параметрите

```
Type[] parameterTypes = constructor.GetParameters();
```

Създаване на нови обекти

- Създаване на нови обекти с помощта на конструктор

```
StringBuilder builder =
  (StringBuilder)constructor.Invoke(new object[] params);
```

Предоставя параметри-обекти за всеки параметър в конструктора, който извикваме

Отражение на методи

- Предоставя публичните методи

```
MethodInfo[] publicMethods = sbType.GetMethods();
```

- Достъп до отделен метод

```
MethodInfo appendMethod =
                    sbType.GetMethod("Append");
MethodInfo overloadMethod = sbType.GetMethod(
                  "Append", new []{typeof(string)});
```

Извикване на метод

- Достъп до параметрите на метод и връщания тип данни

```
ParameterInfo[] appendParameters =
                    appendMethod.GetParameters();
Type returnType = appendMethod.ReturnType;
```

- Извиква методи

```
appendMethod.Invoke(
                builder, new object[] { "hi!“ });
```

Параметри на метода

Екземпляр на обекта-цел

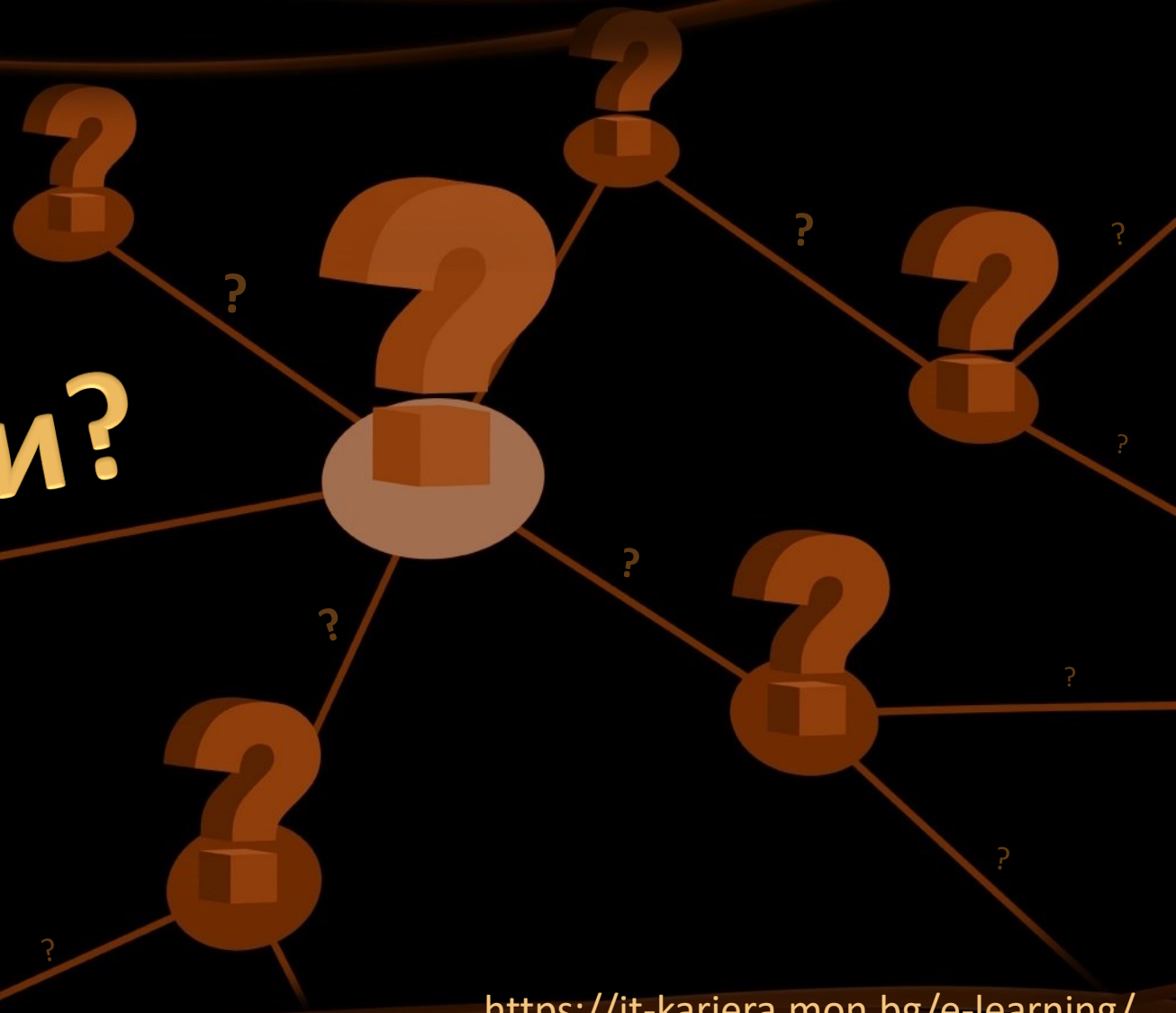
Обобщение

Какви са:

- Ползите
- Негативите на използването на отражения?

Отражение на типовете

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**