Лекция 1
Введение в ООП

Парадигмы программирования

- Парадигма программирования - совокупность принципов идей и понятий которые определяют стиль написания компьютерной программы.

- Императивное программирование
- Структурное программирование
- Функциональное программирование
- Декларативное программирование
- Логическое программирование
- Объектно-ориентированное программирование
- Аспектно-ориентированное программирование

Основные понятия ООП

- **Объект** — это сущность, которой можно посылать сообщения и которая может на них реагировать (методы), используя свои данные (свойства).
- **Абстракция** - выделение значимой информации и исключение из рассмотрения незначимой. В ООП под абстракцией подразумевается набор значимых характеристик объекта, доступный остальной программе.
- **Класс** - модель ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом.
- **Объект** - экземпляр класса.

«3 кита» ООП

- **Инкапсуляция.** Любая программная сущность, обладающая нетривиальным состоянием, должна быть превращена в замкнутую систему, которую можно только перевести из одного корректного состояния в другое. (Разделение интерфейса и сокрытой реализации)
- **Наследование.** Механизм создания новых классов на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.
- **Полиморфизм.** Программная сущность может использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта (динамическая диспетчеризация).

Модификаторы доступа

- **public**: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
- **private**: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.
- **protected**: такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.
- **internal**: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).
- **protected internal**: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.
- **private protected**: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.

Конструкторы

- Кроме обычных методов в классах используются также и специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта. При объявлении конструктора не указывается тип возвращаемого значения, его имя совпадает с именем класса
- Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор по умолчанию. Такой конструктор не имеет параметров и не имеет тела.

Пример

```
class Person
{
    private string _name;
    private decimal _height = 175.0m;

    public Person(string name)
    {
        this._name = name;
    }

    public string SayHello()
    {
        return $“Hello, my name is {name} ";
    }

}
```

Статические элементы

- Статические элементы (свойства и методы) – элементы, принадлежащие классу, а не конкретному экземпляру (объекту)
- Статический класс – класс, обладающий только статическими свойствами и методами. Создать экземпляр статического класса нельзя

  Пример – класс Math
- В некоторых случаях используются статические методы конструирования (различные в зависимости от контекста), использующие приватные конструкторы

Пример

```
class Person
{
    private string _name;
    private decimal _height;
    private const decimal SM_IN_INCH = 2.54m;

    private Person(string name, decimal height)
    {
        this._name = name;
        this._height = height;
    }

    public static CreatePersonInInch(string name, decimal height)
    {
        return new Person(name, height*this.SM_IN_INCH)
    }

          ….

}
```

Наследование

```
sealed class Employee : Person
{
    public string _company;

    public Employee(string name, decimal height, string company) :
       base(name, height)
    {
        _company = company;
    }
}
```

Свойства

```
class Person
{
    private string _name;
    public string Name
    {
        get
        {
            return name;
        }
        private set
        {
            name = value;
        }
    }
}
```

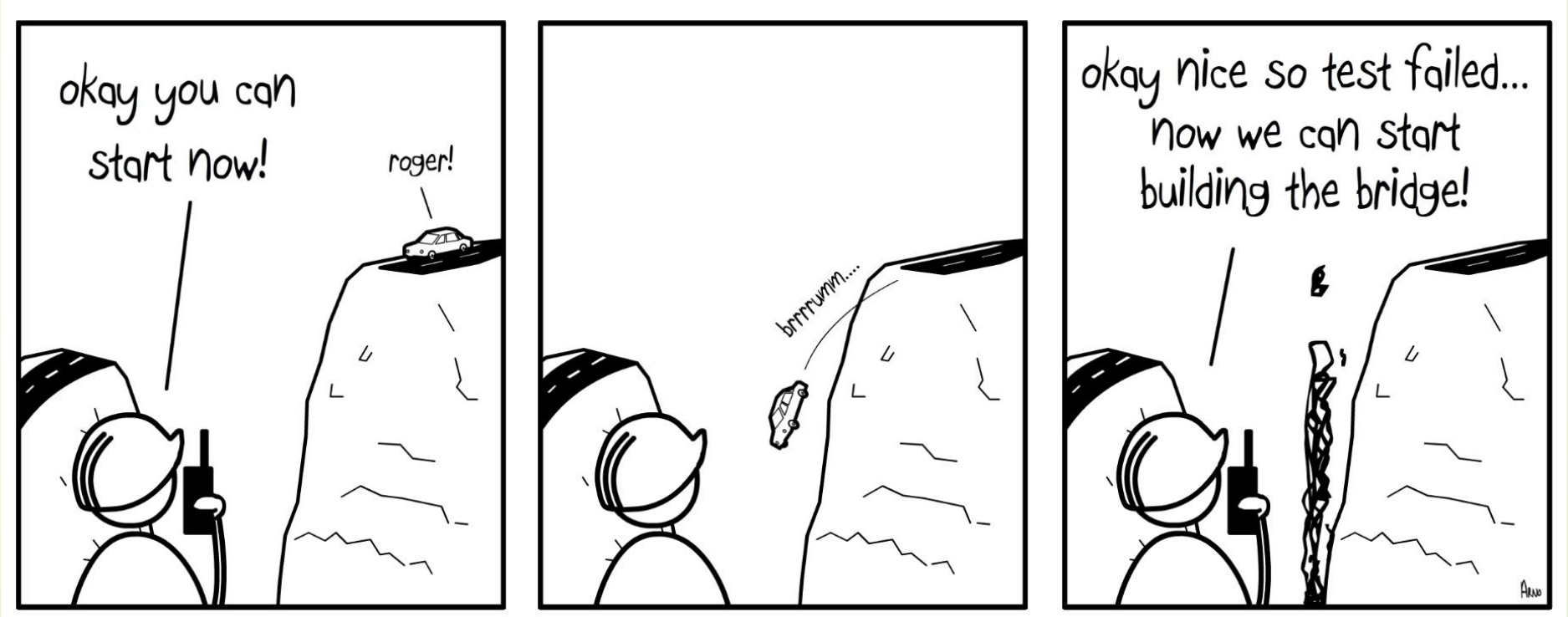
Автоматические свойства

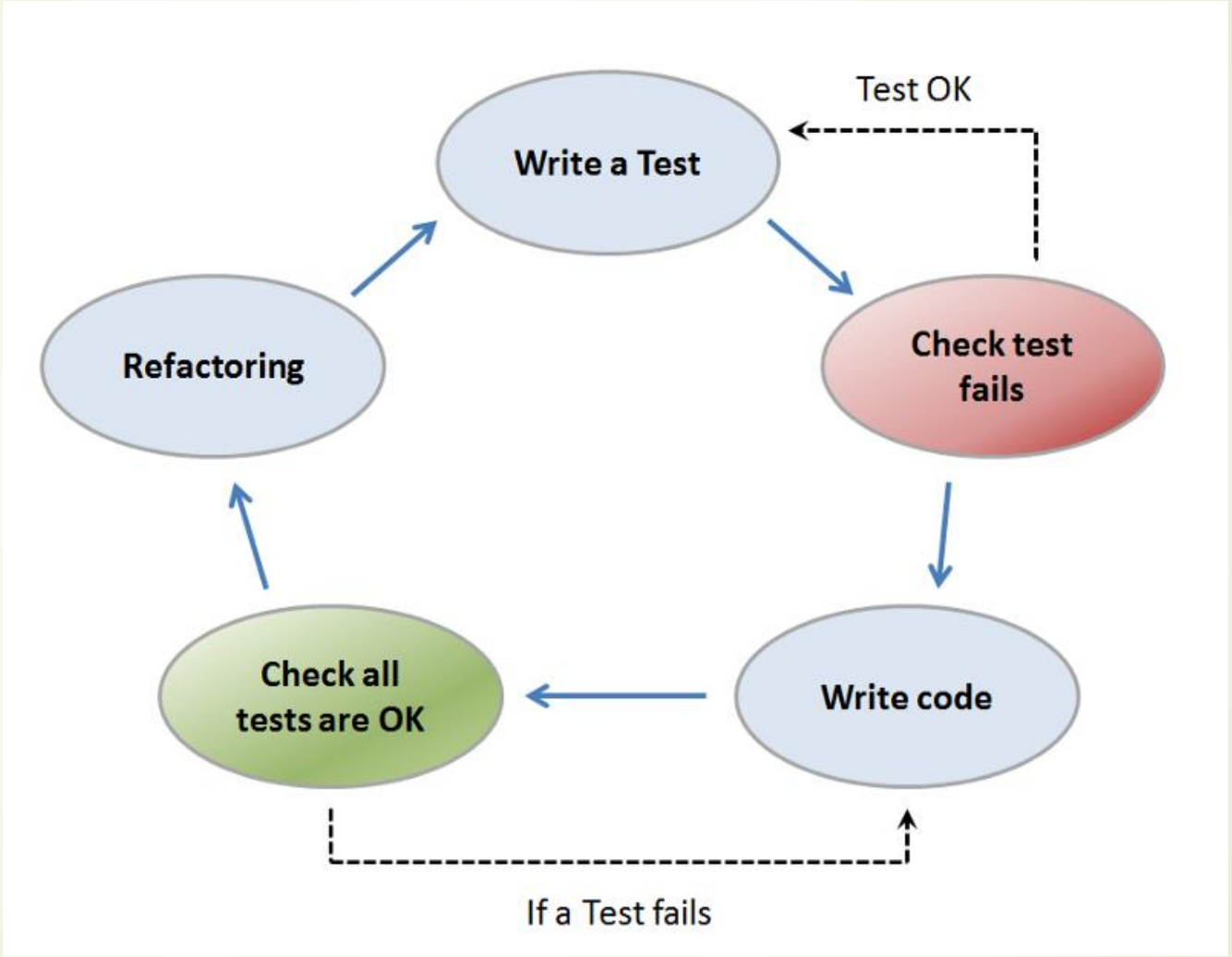
```
class Person
{
    public string Name { get; private set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}
```

Модульные тесты

```
[TestClass]
public class GeometryTests
{
    [TestMethod]
    public void RectangleArea_3and5_15returned()
    {
        var a= 3; var b= 5; var expected = 15;
        var g = new Geometry();
        var actual = g.RectangleArea(a, b);
        Assert.AreEqual(expected, actual);
    }
}
```

Test-driven development (TDD, разработка через тестирование)




Test OK
Write a Test
Check test fails
Write code
Check all tests are OK
Refactoring
If a Test fails