Object oriented programming

Abstractions, Abstract classes, Interfaces

What is “*Abstraction*”

- Focus on some aspects of a system (concepts; “*important stuff*”) while safely ignoring others (“*details*”)
- This is actually related to characteristics of human mind, the way we think
- In OOP the terms abstractions basically means ***interfaces*** or ***abstract classes***

Abstract classes

- Abstract class is a class that cannot be instantiated (objects of this class cannot be created)
- Let’s go back to our *Shapes*

```
class Shape
{
public:
    virtual const double getArea() const
    {
        return 0; // isn’t that useless
    }
    virtual const double getPerimeter() const
    {
        return 0; // unknown shape with perimeter. Really?
    }
};
```

Abstract classes contd. - pure virtual functions

- Actually we don’t need to create any objects of class *Shape*, because polymorphically we use only *concrete* shapes, like *Square, Rectangle, Circle*
- Then *getArea()* and *getPerimeter()* are not only useless, but undefined for objects of class *Shape*
- We can give no implementation (“*noneness*”) to a virtual function by declaring it virtual, but with “*= 0;*” in the end of declaration.
- That types of functions are called ***pure virtual functions***
- ***A class with at least one pure virtual function becomes abstract class and as that it cannot be instantiated!***

Interfaces

- An interface describes behaviour or capabilities of a class, without commiting to a particular implementation (max pureness)
- In C++ there is no difference in implementing interfaces or abstract classes - the difference is only logical
- All said about abstract classes is valid for interfaces too
- By convence when the name of a class finishes with “*able*” it is recognised as interface

Abstractions - why/when to use them ?

- Abstractions are supposed to be designed carefully and be ***more stable*** than details (concretes).
- They are more sustainable to changes
- Thus, more dependable
- ***Rule: Depend on abstractions, not on concrete classes.***