# Praxisorientierte Einführung in C++
# **Aufgabenblatt 9**

Christof Elbrechter
celbrech@techfak.uni-bielefeld.de

18. Juni 2014

## Aufgabe1: Mehrfachvererbung

Gegeben seien folgende Ausgangsdateien: “serializable.h”, “size.h” und “point.h”. Die Dateien finden Sie auch auf der Vorlesungswebseite.

```
#ifndef SERIALIZABLE_H
#define SERIALIZABLE_H

#include <string>
#include <sstream>
#include <iostream>

struct Serializable{
  virtual std::string serialize() const = 0;
};

std::ostream &operator<<(std::ostream &str, const Serializable &s);
```

```
#ifndef POINT_H
#define POINT_H

#include <serializable.h>

struct Point : public Serializable{
  int x,y;
  Point(int x=0, int y=0):x(x),y(y){}
  virtual std::string serialize() const;
};

#endif
```

```
#ifndef SIZE_H
#define SIZE_H

#include <serializable.h>

struct Size : public Serializable{
  int width,height;
  Size(int width=0, int height=0):
    width(width),height(height){}
  virtual std::string serialize() const;
};

#endif
```

a. (3 Punkte) Implementieren Sie die `serialize`-Methoden für die Klassen `Size` und `Point`. Ein `Point` soll nach dem Schema `(x,y)` also z.B. `(3,5)` serialisiert werden; für `Size` soll das Schema `width x height` (ohne Leerzeichen) also z.B. `640x480` verwendet werden. Die zurückgegebenen Strings dürfen keine zusätzlichen Füllzeichen wie *Spaces* oder *Newlines* enthalten. Die Methoden können `inline` definiert werden.

b. (2 Punkte) Implmentieren Sie den `ostream`-Operator

```
std::ostream &operator<<(std::ostream &stream, const Serializable &s);
```

im Header "serializable.h". Der Operator soll das generische Interface von `Serializable` ausnutzen, um beliebige `Serializable`-Typen (auch Sub-Typen) in einen `std::stream` zu *streamen*.

c. (5 Punkte) Implementieren Sie die Klasse `Rect` (in einer Datei "rect.h"), die `Point`, `Size` und nochmals `Serializable` `public` beerbt. Lösen Sie die dabei entstehenden Zweideutigkeiten durch `virtual` Vererbung auf. Implementieren Sie hier erneut die `serialize`-Methode um `Rect`-Instanzen nach dem Schema `Size@Point` also z.B. `640x480@(2,3)` zu serialisieren. Versuchen Sie hierbei die Serialisierungsfunktionen der Oberklassen `Point` und `Size` aufzurufen um deren Funktionalität nicht doppelt implementieren zu müssen.

d. (2 Punkte) Schreiben Sie ein Programm (in einer Datei "main.cpp", welches ein Objekt vom Typ `Rect` konstruiert und auf der Standardausgabe ausgibt.

## Aufgabe2: Exceptions (15 Punkte)

Ausgangspunkt für diese Aufgabe stellt die folgende *Light*-Version einer Image-Klasse dar:

```
#ifndef VGA_IMAGE_H
#define VGA_IMAGE_H
#include <string>
#include <iostream>
#include <fstream>
#include <iterator>
#include <algorithm>
struct VGAImage {
  static const unsigned int W = 640;
  static const unsigned int H = 480;
  static const unsigned int DIM = W*H;

  unsigned char data [DIM];
  inline VGAImage(){
    std::fill(data, data +DIM,0) ;
  }

  inline void save(const std::string & filename){
    std::ofstream os(filename.c_str());
    os << "P2" << '\n' << W << ' ' << H << '\n' << (int)*std::max_element(data,data+DIM) <<
        '\n';
    for(unsigned int y=0;y<H;++y){
      std::copy(data+y*W, data+(y+1)*W,
                std::ostream_iterator<unsigned int>(os," "));
      os << std::endl;
    }
  }

  inline void load(const std::string & filename){
    std::ifstream is(filename.c_str());
    std::string magic_key;
    unsigned int w,h, maxval ;
    is >> magic_key >> w >> h >> maxval;
    std::copy(std::istream_iterator<unsigned int>(is),
              std::istream_iterator<unsigned int>(),data);
  }
  inline unsigned char & operator()(int x, int y) { return data[x+y*W]; }
```

```
  inline const unsigned char & operator()(int x, int y) const { return data[x+y*W]; }
};
# endif
```

a. (3 Punkte) Passen Sie die Load-Funktion in der Art an, dass Sie eine Exception vom Typ

```
InvalidImageSizeException
```

wirft, falls die Größe eines zu ladenen Bildes nicht $640 \times 480$ entspricht. Hierfür müssen Sie natürlich zunächst die Klasse `InvalidImageSizeException` Implemetieren. Diese soll `std::runtime_error` erweitern, welche im Standard-Header `<stdexcept>` zur Verfügung gestellt wird.

b. (3 Punkte) Implementieren Sie ein Interface namens `Drawable`, welches eine rein-virtuelle Methode mit der folgenden Signatur aufweist:

```
virtual void drawTo(VGAImage &image, unsigned char grayValue) const throw (
    DrawException);
```

Hierfür müssen Sie zunächst auch die Klasse `DrawException` implementieren; diese soll ebenfalls von `std::runtime_error` erben.
Beachten Sie dabei, dass bei der Deklaration des `Drawable`-Interfaces die Klasse `VGAImage` noch nicht definiert wurde. Lösen Sie diese Problem mit einer **Vorausdeklaration** der Klasse `VGAImage`.

c. (3 Punkte) Implementieren Sie eine Methode

```
void VGAImage::draw(const Drawable &d, unsigned char grayValue) throw(DrawExecption);
```

als Methode der Klasse `VGAImage`. Diese soll generisch in der Lage sein, Instanzen vom Typ `Drawable` und deren Subtypen in ein Bild zu zeichnen.

d. (3 Punkte) Erweitern sie die Klassen `Point` und `Rect` in der Art, dass Sie das `Drawable`-Interace implementieren. `Point2D::drawTo()` soll den entsprechenden Pixel im Bild mit `grayValue` einfärben. Falls sich der Pixel außerhalb des Bildes befindet, wird eine `DrawException` geworfen. Analog dazu soll `Rect2D::drawTo()` das entsprechende Rechteck im Bild einzeichnen (gefüllt, also nicht nur den Rand).

e. (3 Punkte) Schreiben Sie ein *kleines* Main-Programm (Dateiname: "lena.cpp"), welches ein Bild erstellt und mithilfe von `Rect` und `Point` versucht, etwas in das Bild zu zeichnen, das dem berühmten "Lena"-Testbild ähnlich sieht. Die Gruppe mit dem schönsten Bild erhält 5 Bonus-Punkte. Welches das schönste Bild ist, wird in den Übungen per Abstimmung ermittelt :-)