Eine einfache Klasse für eine Personenbeschreibung hat die folgende UML:

CPerson
private: char Nachname[30] char Vorname[30]
public: CPerson() void setNachname() void setVorname() void getName()

Das Programm:

```
#include<iostream>
#include<string.h>
#include<conio.h>
using namespace std;
class CPerson
{
private:
      char Nachname[30];
      char Vorname[30];
public:
      CPerson();
      void setNachname();
      void setVorname();
      void getName();
};
CPerson::CPerson()    //Standardkonstruktor
{
      strcpy_s(Nachname,"Mustermann");
      strcpy_s(Vorname,"Max");
}
void CPerson::setNachname()
{
      cout << "\nNachname: ";
      cin >> Nachname;
}
void CPerson::setVorname()
{
      cout << "\nVorname: ";
      cin >> Vorname;
}
void CPerson::getName()
{
      cout << endl << Vorname << " " << Nachname << endl;
}
```

```
int main()
{
	CPerson p;
	p.getName();
	p.setNachname();
	p.setVorname();
	p.getName();
	_getch();
	return(0);
}
```

Die Ausführung des Programms könnte folgendermaßen aussehen:

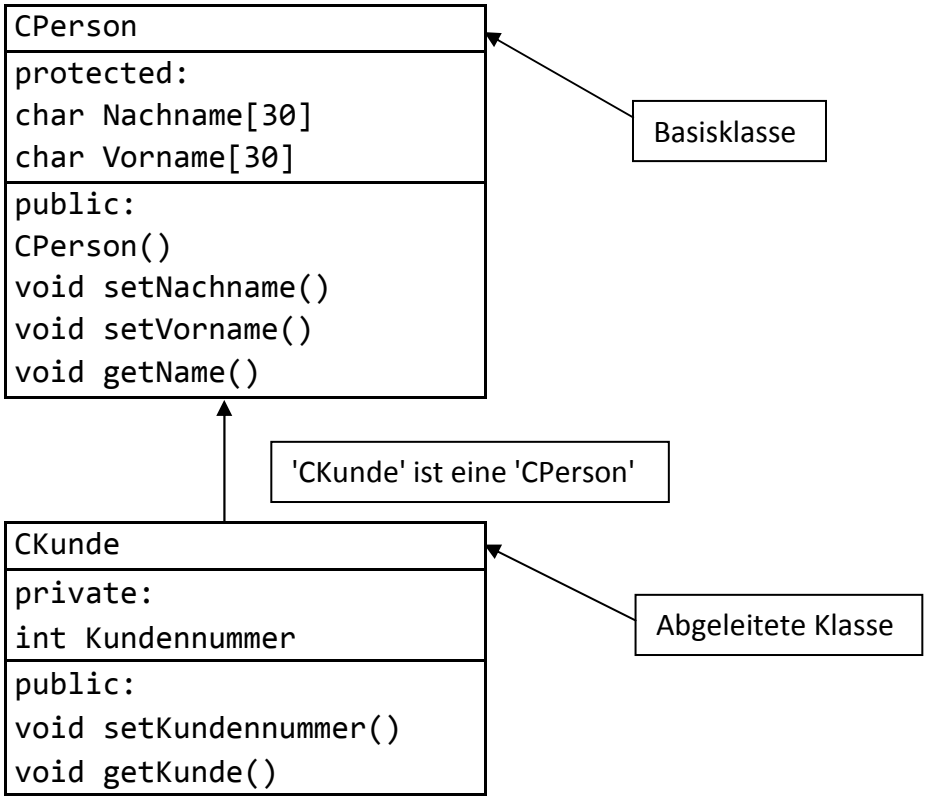
```
Max Mustermann

Nachname: Müller

Vorname: Hans

Hans Müller
```

Die Klasse CPerson beschreibt ganz allgemein die Eigenschaften, welche allen Personen zugeschrieben werden können. In die allgemeine Beschreibung könnte noch die Adresse, der Geburtsort, das Geburtsdatum, die Telefonnummer, usw. aufgenommen werden.

Spezielle Ausprägungen einer Person sind zum Beispiel der Kunde einer Firma, der Mitarbeiter eines Betriebes oder der Student einer Hochschule. Diese speziellen Personen besitzen zusätzliche Eigenschaften und Methoden. Eine Kunde einer Firma hat als zusätzliche Eigenschaft beispielsweise eine Kundennummer, der Mitarbeiter eines Betriebes hat ein gewisses Jahresgehalt, der Student einen Ausbildungsort, usw. .

Allen speziellen Personen sind die Eigenschaften und Methoden der so genannten Basisklasse CPerson gemeinsam. In der objektorientierten Programmierung werden die speziellen Personen in abgeleiteten Klassen beschrieben. Die abgeleiteten Klassen beschreiben nur die zusätzlichen Eigenschaften und Methoden der Basisklasse. Beispielsweise eine abgeleitete Klasse CKunde hat außer Nachname und Vorname die zusätzliche Eigenschaft einer Kundennummer.

Darstellung in UML:



Ein weiteres Beispiel wäre eine Basisklasse Fläche. Rechteck, Kreis, Dreieck, usw. sind spezielle Flächen mit zusätzlichen Eigenschaften und eventuell unterschiedlichen Methoden. Eine abgeleitet Klasse lässt sich begründen, wenn diese sich mit der Umschreibung „…ist ein…“ oder „…hat ein…“ formulieren lässt. Zum Beispiel: „Kunde ist eine Person“ oder „Kreis hat eine Fläche“, usw. .

Eine abgeleitet Klasse wird mit folgender Syntax beschrieben:

```
Class aname : public bname
{
//zusätzliche Eigenschaften und Methoden
…
}
```

In diesem Beispiel ist **aname** der Name der abgeleiteten Klasse und **bname** der Name der Basisklasse. Das Schlüsselwort **public** bewirkt, dass alle öffentlichen Methoden der Basisklasse auch für die abgeleitete Klasse öffentlich sind.

Die Eigenschaften der ursprünglichen Basisklasse sind **private**. Dadurch wäre der Zugriff auf diese Eigenschaften nur mit den Methoden der Basisklasse möglich. Um den abgeleiteten Klassen den Zugriff zu ermöglichen, müssen diese Eigenschaften mit **protected** deklariert werden!

Das vollständige Programm:

```
#include<iostream>
#include<string.h>
#include<conio.h>
using namespace std;
class CPerson
{
protected:
	char Nachname[30];
	char Vorname[30];
public:
	CPerson();
	void setNachname();
	void setVorname();
	void getName();
};
CPerson::CPerson()    //Standardkonstruktor
{
	strcpy_s(Nachname,"Mustermann");
	strcpy_s(Vorname,"Max");
}
void CPerson::setNachname()
{
	cout << "\nNachname: ";
	cin >> Nachname;
}
void CPerson::setVorname()
{
	cout << "\nVorname: ";
	cin >> Vorname;
}
void CPerson::getName()
{
	cout << endl << Vorname << " " << Nachname << endl;
}
class CKunde : public CPerson
{
private:
	int Kundennummer;
public:
	void setKundennummer();
	void getKunde();
};
void CKunde::setKundennummer()
{
	cout << "\nKundennummer: ";
	cin >> Kundennummer;
}
void CKunde::getKunde()
{cout << Kundennummer << " " << Vorname << " " << Nachname << endl;}

int main()
{
	CKunde k;
	k.setNachname();
	k.setVorname();
	k.setKundennummer();
	k.getKunde();
	_getch();
	return(0);
}
```

Im vorigen Beispiel wurde für die Basisklasse ein Standardkonstruktor eingefügt, welcher den Namen Max Mustermann für ein Objekt vergibt. Die abgeleitete Klasse hat die zusätzliche Eigenschaft Kundennummer, welche nach dem Anlegen des Objektes unbestimmt ist.

Beim Anlegen eines Objektes der abgeleiteten Klasse wird grundsätzlich zuerst der Konstruktor der Basisklasse aufgerufen und danach der Konstruktor der abgeleiteten Klasse!

Der Konstruktor der abgeleiteten Klasse braucht lediglich die neuen Eigenschaften zu initialisieren. Alternativ kann er auch den Konstruktor der Basisklasse überschreiben.

Abgeleitete Klasse mit Konstruktor:

```
…
class CKunde : public CPerson
{
private:
      int Kundennummer;
public:
      CKunde();
      void setKundennummer();
      void getKunde();
};
CKunde::CKunde()//Standardkonstruktor
{
      Kundennummer=9999;
}
…
int main()
{
      CKunde k;
      k.getKunde();
      return(0);
}
```

Das Hauptprogramm legt lediglich ein Objekt CKunde an und zeigt die durch den Konstruktor definierten Werte:

```
 9999 Max Mustermann
```

Übung:

Ergänzen Sie das Beispiel um eine abgeleitete Klasse CMitarbeiter mit der zusätzlichen Eigenschaft Jahresgehalt und den Methoden void setJahresgehalt() und void getMitarbeiter()! Die Methode getMitarbeiter() zeigt in einer Zeile neben dem Namen das Jahresgehalt an. Erstellen Sie ebenfalls ein Hauptprogramm zum Testen dieser Klasse!

Vollständige UML: