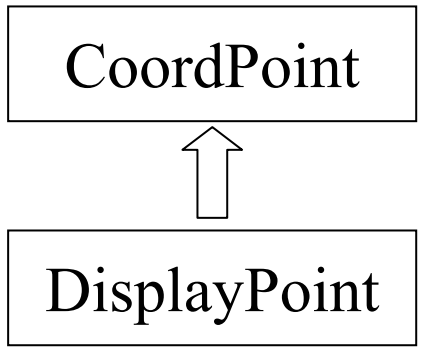
Vienkāršā mantošana. Metožu pārdefinēšana

Superklase un *apakšklase*

(*bāzes* klase un *atvasinātā* klase)



Mantošanas realizācija programmā

```
class DisplayPoint : public CoordPoint {
   private:
      unsigned int Color;
   ...
}
```

Atribūtu sekcija klasē *CoordPoint*

```
class CoordPoint {
   protected:
      int X;
      int Y;
   public:
      ...
}
```

Virtuālās funkcijas klasē *CoordPoint*

```
class CoordPoint {
      ...
   public:
      virtual ~CoordPoint() {
         cout << "Message from the \"CoordPoint\" –
            destroyed!" << endl;
      }
      ...
      virtual void Print() const;
};
```

Virtuālā destruktora efekts

```
CoordPoint  *DP2 = new DisplayPoint();
...
delete DP2;
```

Ziņojums:

```
Message from the "DisplayPoint" - destroyed!
Message from the "CoordPoint" - destroyed!
```

Lai klases *CoordPoint* destruktors **nav virtuālais**. Ziņojums:

```
Message from the "CoordPoint" - destroyed!
```

Nebija izpildīts *apakšklases* destruktors.

Superklases *CoordPoint* konstruktora izsaukums

```
DisplayPoint::DisplayPoint(int Px, int Py,
   unsigned int PColor) : CoordPoint(Px, Py) {
      Color = PColor;
}
```

Metodes *Print()* pārdefinēšana (*overriding*)

```
class CoordPoint {
   ...
   virtual void Print() const;
};

class DisplayPoint : public CoordPoint {
   ...
   virtual void Print() const;
};
```

Metodes *Print()* realizācija apakšklasē

```
inline void DisplayPoint::Print() const {
   CoordPoint::Print();
   cout << ", Color = " << Color;
}
```

Objektu masīvs (klases *CoordPoint*, *DisplayPoint*)

```
CoordPoint *Points[N] = {
   new CoordPoint(1, 2),
   new DisplayPoint(),
   new DisplayPoint(3, 4, 11)
};
```

Objektu saturs (metodes *Print()* izsaukums)

```
Array of points:
1. X = 1, Y = 2                  // Print() no CoordPoint
2. X = 0, Y = 0, Color = 0   // Print() no DisplayPoint
3. X = 3, Y = 4, Color = 11 // Print() no DisplayPoint
```

Metodi *Print()* var arī realizēt pilnīgi pa jauni

```
inline void DisplayPoint::Print() const {
   cout << "X = " << X << ", Y = " << Y <<
      ", Color = " << Color;
}
```

Daži speciālisti nerekomendē izmantot `protected`

Tad apakšklasēs izmanto *superklases piekļuves metodes*

```
class CoordPoint {
  private:
      int X;
      int Y;
.....
inline void DisplayPoint::Print() const {
   cout << "X = " << GetX() << ", Y = " <<
     GetY() << ", Color = " << GetColor();
}
```