Hijerarhijska dekompozicija

- Generalizacijom se uvode osnovne klase koje uopštavaju, apstrahuju, grupišu zajednička svojstva i ponašanje posebnih klasa
- Sama činjenica da neke klase imaju zajednička svojstva ili operacije može, ali ne mora biti dobar razlog za uvođenje njihove generalizacije

```
class Rectangle {
public:
  void draw (Viewport*);
  …
};

class Polyline {
public:
  void draw (Viewport*);
  …
};
```

- Osnovni motiv za generalizaciju jeste to što neka druga apstrakcija ili drugi deo softvera (klijent) ima potrebu da instance datih različitih pojedinačnih klasa *posmatra* i *koristi* na isti način, kroz isti uopšteni *interfejs*, ne praveći razliku između njih
- Na primer, u programu za crtanje dijagrama, na *crtežu* (*Drawing*) se nalaze figure različitih vrsta: pravougaonici (*Rectangle*), poligonalne linije (*Polyline*), krugovi (*Circle*) itd. Kako ih iscrtati?

```
class Circle {
public:
  void draw (Viewport*);
  …
};

void Drawing::draw (Viewport* vp)
{
  // How to draw figures
  // of different kind?
}

…
```

Hijerarhijska dekompozicija

- U nekom proceduralnom jeziku bismo se oslonili na već pokazano, klasično rešenje:

```
switch (fig->kind) {
  case rectangle: drawRectangle(fig,vp);
  case polyline: drawPolyline(fig,vp);
  case circle: drawCircle(fig,vp);
  …
}
```

- Pošto je *Crtež* (*Drawing*) klijent koji različite figure posmatra na isti način, jer sve želi da ih *nacrta* (*draw*), potrebna nam je generalizacija različitih vrsta figura
- Ova generalizacija onda grupiše i zajedničke osobine i zajednički interfejs posebnih klasa:
  - činjenicu da se mogu smeštati na crtež
  - uslugu da se iscrtaju