Klasa kao realizacija softverske mašine

- U nekim slučajevima, reakcija objekta zavisi ne samo od *pobude* (pozvane operacije), nego i od *predistorije* pobuda, odnosno od trenutnog *stanja* tog objekta
- Primer: softver koji upravlja automatom za prodaju napitaka - reakcija na pritisak tastera za izbor artikla zavisi od toga šta se prethodno dogodilo
- Automat može da bude *ispravan* ili u stanju *van upotrebe* (npr. zbog problema ili punog kontejnera za novac) itd.
- Kako bi se ovakvo ponašanje implementiralo? Na primer:

```
void VendingMachine::insertCoin (int value) {
  if (this->inOrder)
    this->amount += value;
  else
    if (!this->isRefundBarrierOpen)
      this->openRefundBarrier();
}

void VendingMachine::selectArticle (int code) {
  if (this->inOrder)
    if (this->articles[code]>0 && this->prices[code]<=this->amount) {
      this->amount -= this->prices[code];
      this->deliverArticle(code);
    }
}

...
```

- Problem: kod složenijeg ponašanja, kod postaje nepregledan, težak za razumevanje, težak za održavanje (izmene, proširenja) i podložan greškama

Klasa kao realizacija softverske mašine

- U ovakvim sistuacijama, za modelovanje ponašanja objekata neke klase koristi se *mašina stanja* (*state machine*) — koncept višeg nivoa apstrakcije koji modeluje ponašanje, uz mogućnost predstavljanja modela dijagramom
- Mašine stanja su napredan softverski koncept koji potiče od konačnih automata, ali dodaje mnoge druge, naprednije koncepte, kao što su hijerarhijska (ugnežđena) stanja, ulazne i izlazne akcije i mnoge druge
- Nažalost, nijedan klasičan OO programski jezik ne podržava mašine stanja: ovakvo ponašanje se mora implementirati ručno, što je naporno i podložno greškama
- Rešenje: generisanje koda iz modela - *softversko inženjerstvo zasnovano na modelima* (*model-based software engineering*)