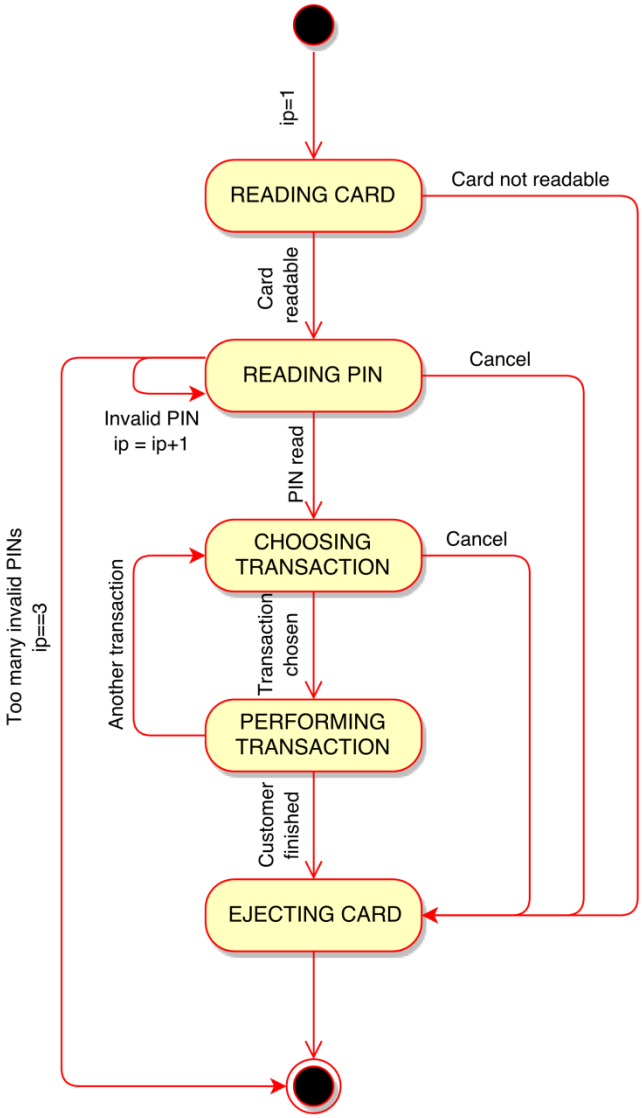
Simulirati rad bankomata čiji je rad opisan sledećim dijagramom stanja i tabelom prelaska iz stanje u stanje.



Stanje u / Stanje iz	READING CARD	READING PIN	CHOOSING TRANSACTION	PERFORMING TRANSACTION	EJECTING CARD	Final
Initial	*Start*, ip = 1 1.0	-	-	-	-	-
READING CARD	-	*Card readable* 0.95	-	-	*Card not readable* 0.05	-
READING PIN	-	*Invalid pin, ip++* 0.05	*PIN read* 0.70	-	Cancel 0.20	*Too many invalid PINs* ip==3
CHOOSING TRANSACTION	-	-	-	*Transaction chosen* 0.80	*Cancel* 0.20	-
PERFORMING TRANSACTION	-	-	*Another transaction* 0.40	-	*Customer finished* 0.60	-
EJECTING CARD	-	-	-	-	-	*Finish* 1.0

Rešenje:

Kod ovog rešenja deklarisane su apstraktne klase stanja i događaja sa čistim vitualnim funkcijama reaguj i akcija, respektivno. Klase stanja (reading_card, pin_read,...) i događaja (card_readable, invalid_pin,...) bankomata konkretizujemo nasleđivanjem ovih apstraknih klasa i realizacijom funkcija reaguj i akcija za svako konkretno stanje i događaj. Dijagram stanja sadrži privatni član poazivača na tekuće stanje (_current). Objekte klasa stanja moguće je dodeliti ovom članu obzirom da su izvedeni iz apstrakne klase stanje (polimorfizam). U funkciji reaguj dijagrama stanja, bez obzira što je tekuce stanje pokazivač na osnovnu klasu stanja, pozvaća se odgovarajuća funkcija reaguj iz izvedenog stanja. Takođe, na sličan način se ostvaruje polimorfizam u funkciji tranzicija prosleđivanjem konstantne reference na objekat događaja.

```
#include <iostream>
#include <ctime>
#include <cstdlib>

using namespace std;

// Funkcija vraca slucajan broj na intervalu od 0 do 1
double random() { return rand()/(1.0*RAND_MAX); }

// Apstraktna klasa stanja
class state {
public:
  // Reakcija na promenu stanja
  virtual void react() = 0;
};

// Apstraktna klasa dogadjaja
class event {
public:
  // Virtuelna funkcija akcije dogadjaja
  virtual void action() = 0;
};

// Izvedeni dogadjaji
// Procitana kartica
class card_readable:public event {
public:
  void action();
};
// Ucitan pin kartice
class pin_read:public event {
public:
  void action();
};
// Izabrana transakcija
class transaction_chosen:public event {
public:
  void action();
};
// Transakcija je zavrsena
class customer_finished:public event {
public:
  void action();
};
// Kartica nije procitana
class card_not_readable:public event {
public:
  void action();
};
// Otkazujem
class cancel:public event {
public:
  void action();
};
// Jos jedna transakcija
class another_transaction:public event {
public:
  void action();
};
```

```
// Pogresan PIN
class invalid_pin:public event {
public:
  void action();
};
// Previse pogresnih PIN-ova
class too_many_invalid_pins:public event {
public:
  void action() {}
};

// Izvedena stanja
// Citanje kartice
class reading_card:public state {
public:
  void react();
};
// Citanje PINa
class reading_pin:public state {
  size_t _ip;
public:
  void react();
};
// Izabiranje transakcije
class choosing_transaction:public state {
public:
  void react();
};
// Transakcija
class performing_transaction:public state {
public:
  void react();
};
// Vracanje kartice
class ejecting_card:public state {
public:
  void react();
};

// Dijagram stanja
class statechart {
  // Zastavica prelaska u zavrsno stanje
  bool _final;
  // Tekuce stanje
  state* _current;
public:
  // Kontruktor
  statechart():_final(false) {}
  // Postavljamo pocetno stanje
  void initial(const state* s) {
    _current = const_cast<state*>(s);
  }
  // Tranzicija iz stanja u stanje za okinuti dogadjaj
  void transition(const event& e) {
    const_cast<event&>(e).action();
  }
public:
  // Postavi u finalno stanje i vrati da li je u finalnom stanju
  bool get_final()                    { return _finish;      }
  void set_final(const bool f)        { _finish = f;         }
  // Postavi i vrati tekuce stanje
  void   set current(const state * s) {  current = const cast<state*>(s); }
  state* get_current() const          { return _current;     }
public:
  // Reakcija dijagrama stanja na promenu stanja
  void react()                        { _current->react();   }
};

// Globalni objekat dijagrama stanja
statechart sc;

// Reakcija sistema na promenu stanja reading_card
void reading_card::react() {
  double p = random();
  if(p<=0.05) {
    sc.transition(card_not_readable());
  }
  else {
```

```
    sc.transition(card_readable());
  }
}
// Reakcija sistema na promenu stanja reading_pin
void reading_pin::react() {
  double p = random();
  if(p<=0.20) {
    sc.transition(cancel());
  }
  else if(p<=0.25) {
    _ip++;
    if( ip<3)
      sc.transition(invalid_pin());
    else {
      sc.transition(too_many_invalid_pins());
      sc.set_finish(true);
    }
  }
  else {
    sc.transition(pin_read());
  }
}
// Reakcija sistema na promenu stanja choosing_transaction
void choosing transaction::react() {
  double p = random();
  if(p<=0.20) {
    sc.transition(cancel());
  }
  else {
    sc.transition(transaction_chosen());
  }
}
// Reakcija sistema na promenu stanja performing_transaction
void performing transaction::react() {
  double p = random();
  if(p<=0.40) {
    sc.transition(another_transaction());
  }
  else {
    sc.transition(customer_finished());
  }
}
// Reakcija sistema na promenu stanja ejecting_card
void ejecting_card::react() {
  sc.set_finish(true);
}

// Globalni objekti stanja dijagrama stanja
// Stanje citanja kartice
reading_card           rc;
// Stanje citanja pina
reading pin            rp;
// Stanje izbora transakcije
choosing_transaction   ct;
// Stanje izvrsenja transakcije
performing transaction pt;
// Stanje izbacivanja kartice
ejecting_card          ec;

// Akcija dogadjaja card_readable
void card_readable::action() {
  if(sc.get current() == &rc) {
    sc.set current(&rp);
    cout << "READING_PIN" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}
// Akcija dogadjaja card_not_readable
void card_not_readable::action() {
  if(sc.get_current() == &rc) {
    sc.set current(&ec);
    cout << "EJECTING_CARD" << endl;
  }
  else {
    cerr << "Error" << std::endl;
```

```
    exit(1);
  }
}
// Akcija dogadjaja pin_read
void pin_read::action() {
  if(sc.get_current() == &rp) {
    sc.set_current(&ct);
    cout << "CHOOSING_TRANSACTION" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}
// Akcija dogadjaja invalid_pin
void invalid_pin::action() {
  if(sc.get_current() == &rp) {
    sc.set_current(&rp);
    cout << "READING_PIN" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}
// Akcija dogadjaja cancel
void cancel::action() {
  if(sc.get_current() == &rp || sc.get_current() == &ct) {
    sc.set_current(&ec);
    cout << "EJECTING_CARD" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}
// Akcija dogadjaja transaction_chosen
void transaction_chosen::action() {
  if(sc.get_current() == &ct) {
    sc.set_current(&pt);
    cout << "PERFORMING_TRANSACTION" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}
// Akcija dogadjaja another_transaction
void another_transaction::action() {
  if(sc.get_current() == &pt) {
    sc.set_current(&ct);
    cout << "CHOOSING_TRANSACTION" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}
// Akcija dogadjaja customer_finished
void customer_finished::action() {
  if(sc.get_current() == &pt) {
    sc.set_current(&ec);
    cout << "EJECTING_CARD" << endl;
  }
  else {
    cerr << "Error" << std::endl;
    exit(1);
  }
}

int main() {
  // Seme generatora slucajnih brojeva iniciramo vremenom.
  // Na ovaj nacin sa svakim izvresenjem programa dobijamo druge vrednosti stanja.
  srand(time(NULL));
  // Postavljamo pocetno stanje
  sc.initial(&rc);
  cout << "READING_CARD" << endl;
  // Prolazimo kroz sva stanja dijagrama stanja dok ne dodjemo do finalnog
```

```
  while(!sc.get_final()) {
    sc.react();
  }
  return 0;
}
```