Behavioural Design Patterns

Part 1

Behavioural Design Patterns

- Concerned with:
  - Algorithms
  - The assignment of responsibilities between objects
- Two types:
  - Class Behavioural - Use inheritance to distribute behaviour between classes
  - Object Behavioural - Use object composition rather than inheritance

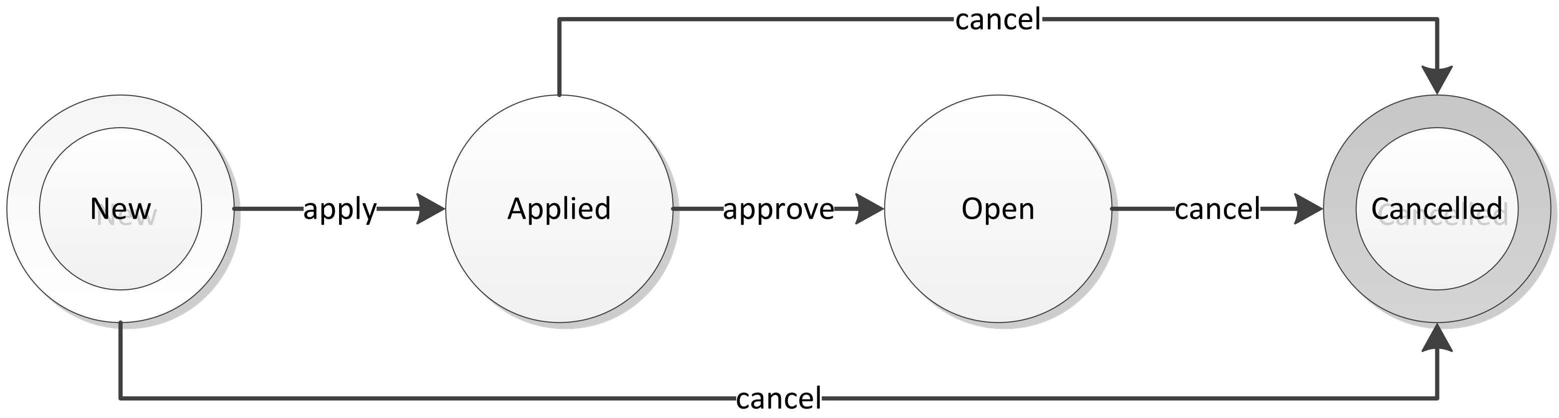
Behavioural Design Patterns

- State
- Strategy
- Observer
- Command
- Visitor

Behavioural Patterns: State

- Suppose we are building a `LineOfCredit` class
- A line of credit can be in various states:
  - New
  - Applied
  - Open
  - Cancelled
- A line of credit has various behaviours:
  - `apply`
  - `withdraw`
  - `makePayment`
  - `cancel`
- Behaviours may change depending on the current state

Behavioural Patterns: State



Behavioural Patterns: State

LineOfCredit.h

```
class LineOfCredit
{
  public:
    enum AccountState { NEW, APPLIED, OPEN, CANCELLED };

    LineOfCredit();

    const std::string state() const;
    float balanceOwing() const;
    float availableCredit() const;

    void apply(float amount);
    void approve();
    void withdraw(float amount);
    void makePayment(float amount);
    void cancel();

  private:
    AccountState _state;
    float _availableCredit;
    float _balanceOwing;
};
```

Behavioural Patterns: State

LineOfCredit.cpp

```
LineOfCredit::LineOfCredit()
{
  this->_state = NEW;
}

const string LineOfCredit::state() const
{
  switch (this->_state)
  {
    case NEW:
      return "New";
    case APPLIED:
      return "Applied";
    case OPEN:
      return "Open";
    case CANCELLED:
      return "Cancelled";
    default:
      return "Unknown";
  }
}
```

Behavioural Patterns: State

LineOfCredit.cpp

```
void LineOfCredit::apply(float amount)
{
  if (this->_state == NEW)
  {
    this->_state = APPLIED;
    this->_availableCredit = amount;
  }
  else
    throw "Can't apply in the current state";
}
```

Behavioural Patterns: State

LineOfCredit.cpp

```
void LineOfCredit::cancel()
{
  switch (this->_state)
  {
    case NEW:
    case APPLIED:
      this->_state = CANCELLED;
      break;

    case OPEN:
      if (this->_balanceOwing > 0)
        throw "If only life worked that way.";
      else
        this->_state = CANCELLED;
      break;

    default:
      throw "Can't cancel the line of credit in the current state";
      break;
  }
}
```

Behavioural Patterns: State

Design Pattern:

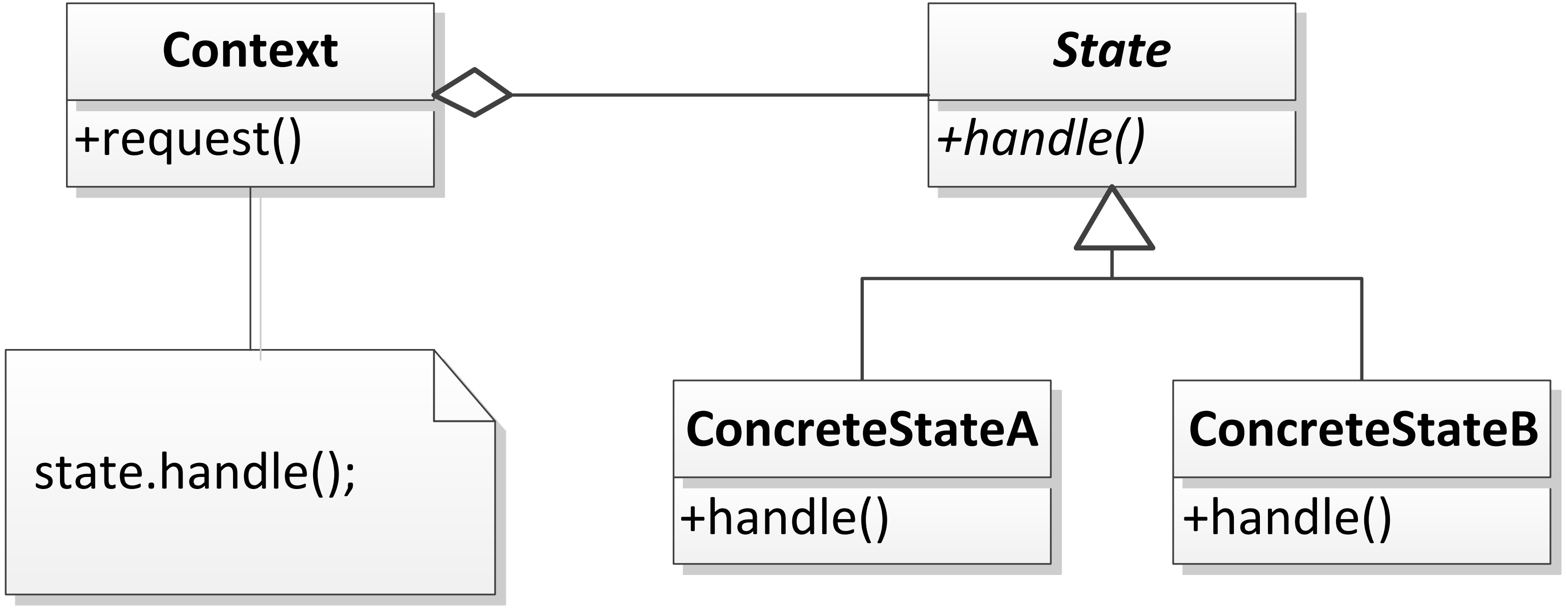
State

Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class.

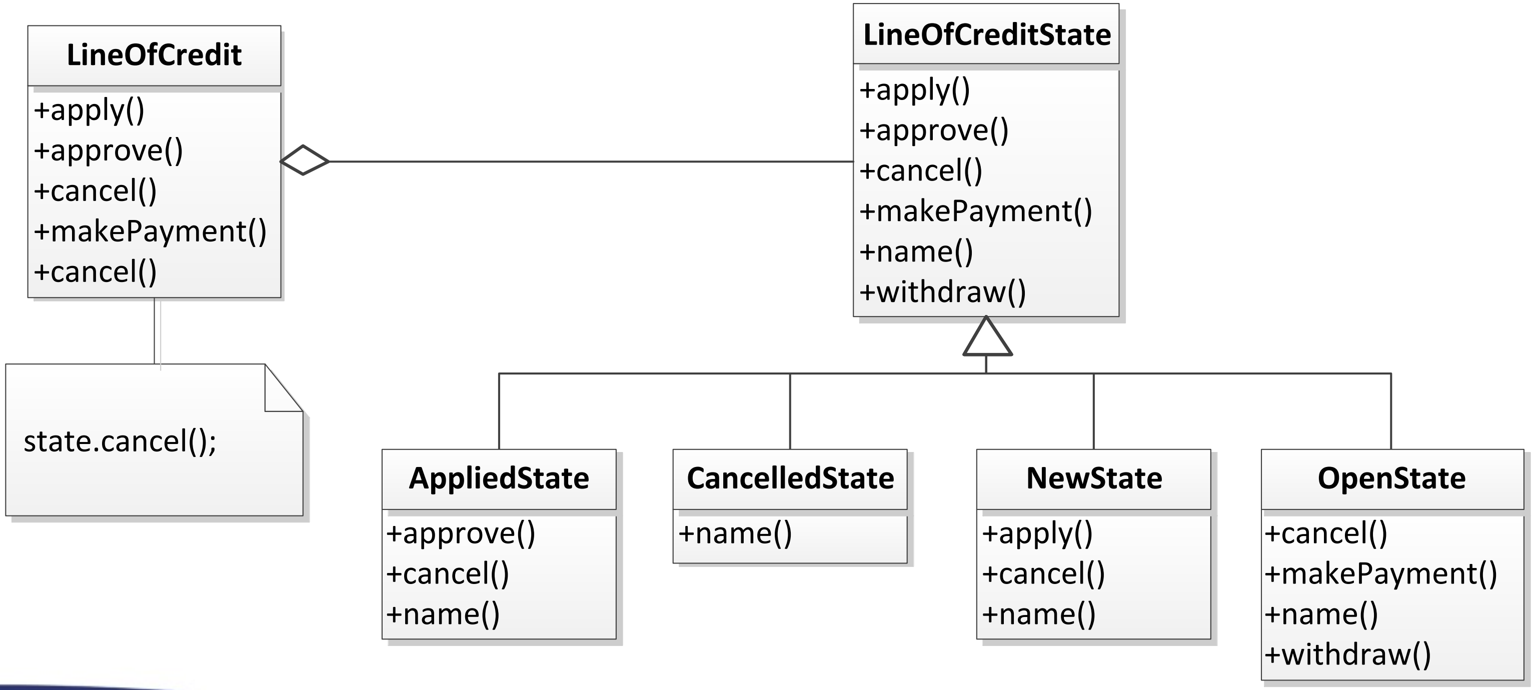
Behavioural Patterns: State

- Applicability:
  - An object’s behaviour depends on its state, and it must change its behaviour at run-time depending on that state
  - Operations have large, multipart conditional statements that depend on the object’s state
    - Usually represented by one or more enumerated constants
    - Often, several operations will contain this same conditional structure

Behavioural Patterns: State



Behavioural Patterns: State



Behavioural Patterns: State

LineOfCredit.h

```
public:

    friend class LineOfCreditState;

    LineOfCredit();
    ...

private:
    LineOfCreditState* _state;
    float _availableCredit;
    float _balanceOwing;
```

Behavioural Patterns: State

LineOfCredit.cpp

```
LineOfCredit::LineOfCredit()
{
  this->_state = new NewState(this);
}

const string LineOfCredit::state() const
{
  return this->_state->name();
}
```

Behavioural Patterns: State

LineOfCredit.cpp

```
void LineOfCredit::apply(float amount)
{
  this->_state->apply(amount);
}

void LineOfCredit::approve()
{
  this->_state->approve();
}

void LineOfCredit::withdraw(float amount)
{
  this->_state->withdraw(amount);
}

void LineOfCredit::makePayment(float amount)
{
  this->_state->makePayment(amount);
}
```

Behavioural Patterns: State

LineOfCreditState.h

```
class LineOfCreditState
{
  public:
    LineOfCreditState(LineOfCredit*);
    virtual void apply(float);
    virtual void approve();
    virtual void withdraw(float);
    virtual void makePayment(float);
    virtual void cancel();
    virtual const std::string name() const;

  protected:
    LineOfCredit* _loc;
};
```

Behavioural Patterns: State

LineOfCreditState.cpp

```
LineOfCreditState::LineOfCreditState(LineOfCredit* loc) : _loc(loc)
{
}

void LineOfCreditState::apply(float amount)
{
  throw "Cannot apply in the current state";
}

void LineOfCreditState::approve()
{
  throw "Cannot approve line of credit in the current state";
}

void LineOfCreditState::withdraw(float amount)
{
  throw "Cannot withdraw from line of credit in the current state";
}
```

Behavioural Patterns: State

AppliedState.cpp

```
AppliedState::AppliedState(LineOfCredit* loc) : LineOfCreditState(loc)
{
}

void AppliedState::approve()
{
  this->_loc->_state = new OpenState(this->_loc);
}

void AppliedState::cancel()
{
  this->_loc->_state = new CancelledState;
}

const string AppliedState::name() const
{
  return "Applied";
}
```

Behavioural Patterns: State

OpenState.cpp

```
OpenState::OpenState(LineOfCredit* loc) : LineOfCreditState(loc)
{
}

void OpenState::withdraw(float amount)
{
  if (this->_loc->_balanceOwing + amount > this->_loc->_availableCredit)
    throw "Insufficient funds available";
  else
    this->_loc->_balanceOwing += amount;
}

void OpenState::makePayment(float amount)
{
  this->_loc->_balanceOwing -= amount;
}
```

Behavioural Patterns: State

OpenState.cpp

```
void OpenState::cancel()
{
  if (this->_loc->_balanceOwing > 0)
    throw "If only life worked that way.";
  else
    this->_loc->_state = new CancelledState;
}

const string OpenState::name() const
{
  return "Open";
}
```

Behavioural Patterns: State

- Consequences:
  - Localizes state-specific behaviour and partitions behaviour for different states
  - Makes state transitions explicit
  - State objects can be shared