# Easy JSON printing

Just a Simple Object-oriented Natural (JSON) Printer for C++

J. Snoeijer

## Outline

- JSON Observations
- Implementation
- Using the JSON Printer in your own projects
- Conclusion

# JSON Observations

```json
{
  "customer": {
    "companyname": "My Company",
    "address": "Mainstreet 123",
    "city": "Acity"
  },
  "paid": 1,
  "lines": [
    {
      "product": "Butter",
      "amount": 1,
      "price": 0.99
    },
    {
      "product": "Cheese",
      "amount": 1,
      "price": 5.39
    }
  ]
}
```

## JSON Observations

```json
{
  "customer": {
    "companyname": "My Company",
    "address": "Mainstreet 123",
    "city": "Acity"
  },
  "paid": 1,
  "lines": [
    {
      "product": "Butter",
      "amount": 1,
      "price": 0.99
    },
    {
      "product": "Cheese",
      "amount": 1,
      "price": 5.39
    }
  ]
}
```

Text-valued property

# JSON Observations

```
{
  "customer": {
    "companyname": "My Company",
    "address": "Mainstreet 123",
    "city": "Acity"
  },
  "paid": 1,
  "lines": [
    {
      "product": "Butter",
      "amount": 1,
      "price": 0.99
    },
    {
      "product": "Cheese",
      "amount": 1,
      "price": 5.39
    }
  ]
}
```

Text-valued property

Numerical-valued property

# JSON Observations

```
{
  "customer": {
    "companyname": "My Company",
    "address": "Mainstreet 123",
    "city": "Acity"
  },
  "paid": 1,
  "lines": [
    {
      "product": "Butter",
      "amount": 1,
      "price": 0.99
    },
    {
      "product": "Cheese",
      "amount": 1,
      "price": 5.39
    }
  ]
}
```

Text-valued property

Object

Numerical-valued property

# JSON Observations

```
{
  "customer": {
    "companyname": "My Company",
    "address": "Mainstreet 123",
    "city": "Acity"
  },
  "paid": 1,
  "lines": [
    {
      "product": "Butter",
      "amount": 1,
      "price": 0.99
    },
    {
      "product": "Cheese",
      "amount": 1,
      "price": 5.39
    }
  ]
}
```

Text-valued property

List property

Object

Numerical-valued property

# JSON Observations

```
{
  "customer": {
    "companyname": "My Company",
    "address": "Mainstreet 123",
    "city": "Acity"
  },
  "paid": 1,
  "lines": [
    {
      "product": "Butter",
      "amount": 1,
      "price": 0.99
    },
    {
      "product": "Cheese",
      "amount": 1,
      "price": 5.39
    }
  ]
}
```

Object property

Text-valued property

List property

Object

Numerical-valued property

## Implementation

Some highlights about the JSON Printer itself

# Implementation - printing basics

- A *Line* is
  - A *LineSegment*, which is
    - Indentation
    - Key (String valued)
    - Value  (Any type)
  - A *LineEnd*

# Implementation - printing basics

- A *Line* is
  - A *LineSegment*, which is
    - Indentation
    - Key (String valued)
    - Value  (Any type)
  - A *LineEnd*

```
template<typename ValueType>
void JSONPrinter::printLineSegment(string key, ValueType val)
{
    printIndentation();

    // print the line to the output file
    output_ << "\"" << key << "\": " << val;
}
```

```
template<typename ValueType>
void JSONPrinter::printLinePlain(string key, ValueType val, bool printEndingComma)
{
    printLineSegment(key, val);
    printEndLine(printEndingComma);
}
```

# Implementation - printing lists

```
template<typename ValueType>
void JSONPrinter::printList(string key, list<ValueType> vals, bool printEndingComma)
{
    if (vals.size() > 0)
    {
        // Print the opening bracket of the list
        printLine<char>(key, '[', false);
        ++indentationLevel_;

        // Print first to one before last element ending with a comma
        typename std::list<ValueType>::iterator lastElement = prev(vals.end());
        for_each(vals.begin(), lastElement, [this](ValueType val) { print(val, true); });

        // Print the last element without an ending comma
        print(*lastElement, false);

        // Print the closing bracket of the list
        --indentationLevel_;
        printIndentation();
        output_ << "]";
        printEndLine(printEndingComma);
    }
    else
        // Print empty list
        printLine<string>(key, "[]", printEndingComma);
}
```

# Implementation - printing objects

```
template<typename DomainSpecificType>
void JSONPrinter::print(DomainSpecificType& object, bool printEndingComma)
{
    // Print begin bracket of object
    printObjectHeader();

    // Sends the object to the DomainExpert such that he prints this domain specific object
    DomainExpert::print(*this, object);

    // Print end bracket of object
    printObjectFooter(printEndingComma);
}
```

A *Domain expert* should know how to print a *Domain specific object*

```
class DomainExpert
{
public:
    static void print(JSONPrinter& printer, const SalesOrder& salesOrder);
    static void print(JSONPrinter& printer, const SalesOrderLine& salesOrderLine);
    static void print(JSONPrinter& printer, const Customer& customer);
};
```

## Usage and Setup

Using the JSON Printer in your own projects

# Usage – encode your objects

```
void DomainExpert::print(JSONPrinter& printer, const SalesOrder& salesOrder)
{
    printer.printLineObject<Customer>("customer", salesOrder.customer_);
    printer.printLine<bool>("paid", salesOrder.paid_);

    // We are at the end of an object and don't want an ending comma at the last line,
    // so set the parameter printEndingComma = false (default = true)
    printer.printList<SalesOrderLine>("lines", salesOrder.lines_, false);
}

void DomainExpert::print(JSONPrinter& printer, const Customer& customer)
{
    printer.printLineQuoted<string>("companyname", customer.companyName_);
    printer.printLineQuoted<string>("address", customer.address_);
    printer.printLineQuoted<string>("city", customer.city_, false);
}
```

## Usage – encode your objects

- JSON Printer has out of the box support for
  - Printing lines with numbers without quotes
  - Printing lines with string with quotes
  - Printing lines with own defined objects

  - Printing general lists
  - Printing general object

```
printLinePlain<float>("pi", 3.1415);
printLineQuoted<string>("key", "Hello World");
printLineObject<MyClass>("class", myObject);

printList<int>("numbers", listOfIntegers);
print<MyClass>(myObject);
```

## Usage – print your first object

- Create an JSONPrinter object with an output …
- … and call the print() function

```
void writeJSONtoStdOut()
{
    // Construct the JSON cout printer.
    JSONPrinter printer(cout);

    SalesOrder order = createSalesOrder();

    // Send the data to the JSON Printer.
    printer.print(order);
}
```

## Conclusion

- JSON Printer provides out of the box standard functions for basic JSON constructs
- Using the *DomainExpert* you can specify how the *JSONPrinter* prints your objects
- Just a Simple Object-oriented Natural (JSON) Printer

JSONPrinter and a running example on GitHub: https://github.com/jsnoeijer/JSONPrinter