UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS 214 Tutorial 1

- This tutorial takes place on **23 August 2021**.
- This tutorial consists of **10 questions**.
- The tutorial does not contribute towards your final marks.

Part I

Question 1 ....(7 marks)

For each of the questions that follow, choose the most appropriate option.

1.1 What is the output of the following code? (1)

```
int x=3;
if(x = 2)
cout << "x = " << x;
else
cout << "The value of x is " << x;

```

A. x = 2

B. x = 3

C. The value of x is 3

D. The value of x is 2

1.2 What will be displayed on the screen when the following code executes? (1)

```
int num = 125;
int *numPtr;

numPtr = &num;
cout <<  numPtr  << '\t'  << *numPtr;

```

A. memory address of variable numPtr, followed by a tab, followed by 125

B. memory address of variable num, followed by a tab, followed by 125

C. value of num in hexadecimal followed by a tab followed by 125

D. memory address of variable numPtr followed by a tab followed by NULL

1.3 What will the output of the following code be? (1)

```
int a[] = {5,6,7,8};
int *b = new int[5];
int *c = b;

if (a != c)
cout << "The arrays are not the same.";
cout << "The arrays are the same.";

```

A. A compilation error.

B. The arrays are the same.

C. The arrays are not the same.The arrays are the same.

D. The arrays are not the same.
The arrays are the same.

E. The arrays are not the same.

1.4 What will `strlen` on `st1` return after the following code executes? (1)

```
const int size =20;
char st1[size] = "Hostess Snack ";
char st2[] = "Cakes";
strcat(st1 ,st2);

```

A. 20

B. 5

C. 19

D. 12

1.5 Which of the following class function prototypes ensures that the `getPrice` function does not modify any of its class's member variables? (1)

A. `const double getPrice();`

B. `double const getPrice();`

C. `double getPrice() const;`

D. `double getPrice(); const`

1.6 What is the default access for members of the struct abstract data type? (1)

A. public

B. protected

C. private

1.7 What is the default access for members of the class abstract data type? (1)

A. public

B. protected

C. private

Question 2 .... (10 marks)

2.1 Consider the following function. (1)

```
bool isSecret(char* s, int len) {
if(len < 2)
return true;
else
return s[0] == s[len-1] && isSecret(&s[1], len-2);
}

```

a) What does the function determine? (1)

> **Solution:** Whether `s` is a palindrome or not.

b) Is this an efficient solution to the problem? (2)

> **Solution:** No, an iterative solution makes use of less memory, especially on the runtime stack.

2.2 Write a function in C++ to solve the following mathematical function, $A(m, n)$, where $m, n \geq 0$. (6)

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases} \tag{1}$$

Note, the function grows really quickly and therefore parameter and return types of `int` will not do. Refer to the examples below:

```
A(0,0) = 1
A(1,0) = 2
A(1,1) = 3
A(2,1) = 3
A(3,1) = 13
A(4,1) = 65533
```

In many cases, the program segfaults before completion due to a stack overflow.

Solution: This is the Ackermann-Péter function.

```cpp
unsigned long long A(unsigned long long m, unsigned long long n) {
// 1 mark for not using int as type
// 1 mark for accepting 2 parameters

if (m == 0) {  // 1 mark for condition and statement
return n + 1;
}
if (n == 0) {  // 1 mark for condition and statement
return A(m - 1, 1);
}
return A(m - 1, A(m, n - 1)); // 1 mark for statement
}
// 1 mark if statements are returned

```

Question 3 .... (18 marks)

Consider the following class definition and answer the questions that follow.

```cpp
template <typename T>
class Matrix {
public:
Matrix(T** matrix, int m, int n);
T findLargest();
void printMatrix();
~Matrix();
private:
Matrix();
T** matrix;
int m;
int n;
};
```

3.1 Write code to initialise a matrix that can be accepted by the `Matrix` class constructor and initialise it with the following values: (6)

```
0 1 2
1 2 3
2 3 4
```

Solution:

```
    rows = 3; // 1 mark - initialising a variable or constant
    cols = 3;

    double** matrix = new double*[rows];
    // 1 mark - creating an array of double pointers
    for (int i = 0; i < rows; i++) {
    // 1 mark - correct for loop
    matrix[i] = new double[cols];
    // 1 mark - creating arrays for each row of column length
    for (int j = 0; j < cols; j++)
    // 1 mark - correct for loop
    matrix[i][j] = j+i;
    // 1 mark = assignment
    }
```

3.2 The `printMatrix` function is implemented as follows:

```
template <typename T>
void Matrix<T>::printMatrix() {
for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++)
cout << matrix[i][j] << "␣";
cout << endl;
}
}
```

Provide an implementation for each of the following functions.

a) `findlargest` (6)

Solution:

```
    template <typename T>
    T Matrix<T>::findLargest() { // 1 mark - correct signature

    T largest = matrix[0][0];   // 1 mark - initialising largest

    for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)  // 1 mark - nested for
    if (matrix[i][j] > largest) // 1 mark - comparison
    largest = matrix[i][j]; // 1 mark - update
```

```
      return largest;  // 1 mark - return largest
      }

```

b) `destructor` (4)

Solution:

```
      template <typename T>
      Matrix<T>::~Matrix() { // 0.5 marks - signature
      for (int i = 0; i < m; i++)  // 1 mark - for loop for number of rows
      delete [] matrix[i]; // 1 mark - deleting a row
      delete [] matrix; // 1 mark - deleting array of pointers
      matrix = nullptr; // 0.5 marks - null for matrix
      }

```

3.3 Why has the default constructor been defined as private? (2)

Solution: To ensure that an object of the Matrix class cannot be instantiated and used before a matrix array has been created and assigned to class attribute.

Question 4 ....(5 marks)

Assume that a class `Data` exists and is referenced by the `Node` class. Provide the definition of a `Node` class that can be used in the implementation of:

4.1 A singly linked list (2)

Solution:

```
     class Node {
     public:
     // The necessary constructors and destructor
     // The necessary insert, remove and get
     private:
     Data data;
     Node* next;  // 1 mark, the other mark for the rest being correct
     };

```

4.2 A binary tree (3)

Solution:

```
     class Node {
     public:
     // The necessary constructors and destructor
     // The necessary insert, remove and get
     private:
     Data data;
     Node* left; // Could be Node* child[2];
     Node* right;
```

```
    // 1 mark per Node and 1 mark for the rest.
    };

```

Part II

You have been approached by a local private brewery to analyse their current production system and propose a design to automate the process.

The brewery produces different types of beer which they then case and crate for shipping. A case comprises of 24 units (bottles). To fill a case, a keg of beer has be be brewed. Once the brewing process has completed, 24 bottles are filled with beer from the keg. A cap is placed on each bottle, and finally so is a label. Depending on the type of beer (Lager, Pilsner or Stout for now) and brand (Hansa, Castle, etc...), different types of caps and labels are placed on the bottles.

Each case of beer is stored by adding it to a crate. A crate has been filled when it contains 32 cases of beer. Once a crate has been filled it is shipped to its destination.

The design of the process is broken down into the following aspects:

- Identification and design, modelling the beer that is brewed (product)
- The design of beer production process
- Storage and shipping of the beer

Question 5 .... (16 marks)

A base class for the product hierarchy is given.

```
class Beer {
public:
  Beer(string, string);
  string getType();
  string getBrand();
private:
  string type;
  string brand;
};
```

5.1 Implement the functions of the class. The parameters of the constructor are the beer type in text and the brand respectively. The values are assigned to the class attributes. The *getters* return the respective strings. (7)

Solution:

```
Beer::Beer(string beerType, string beerBrand){
  type = beerType;
  brand = beerBrand;
}

string Beer::getType() {
  return type;
}
string Beer::getBrand(){
  return brand;
}
```

5.2 Would it be wise to include a virtual destructor in the `Beer` class? Provide a reason for your answer. (2)

Solution: Yes it would. If one of the subclasses allocate heap memory, to ensure that it is deallocated, a virtual destructor in the base class is required. We will be creating a pointer to the base class and instantiating an object of the derived class and assigning it to the pointer. When the object is deleted, it is necessary to ensure that the chain of destructors are called and this is done by making the base class destructor virtual.

5.3 The constructor of each of the derived classes, takes one string parameter – the brand. Provide a definition of the class for one of the derived classed and the implementation of the constructor. (4)

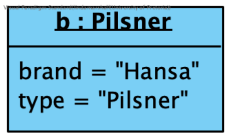
Solution:

```
class Pilsner : public Beer {
public:
  Pilsner(string);
};

Pilsner::Pilsner(string brand) : Beer("Pilsner",brand) { }
```

5.4 For the derived class you chose in the previous question, show how you would instantiate an object of this class and provide a UML Object diagram for the object immediately after instantiation. (3)

Solution: Beer∗ b = **new** Pilsner("Hansa");



Question 6 ....(3 marks)

Once the beer has been cased, it is sent to storage where it is packed into a crate. A crate is able to hold 32 cases of beer. Once a crate is full, it is taken out of storage and shipped. The next empty crate is placed into storage and as the cases are produced, added to the crate. And so the cycle continues.

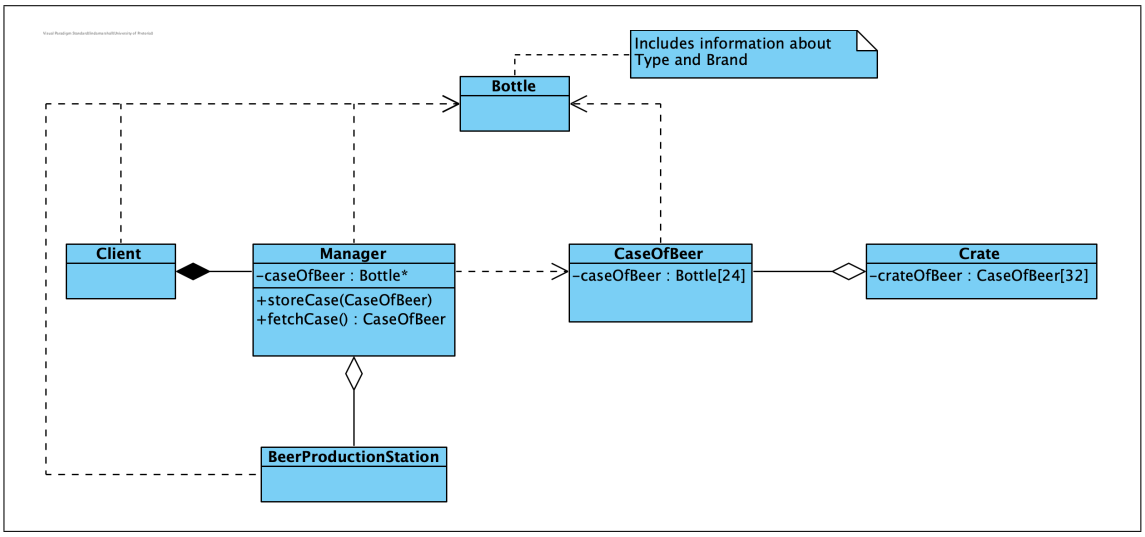
6.1 Which pattern is best suited to manage the filling of the crate? (1)

Solution: Memento

6.2 Do any of the classes already in the design fulfil the roles of participants for the pattern identified in the previous question? (2)

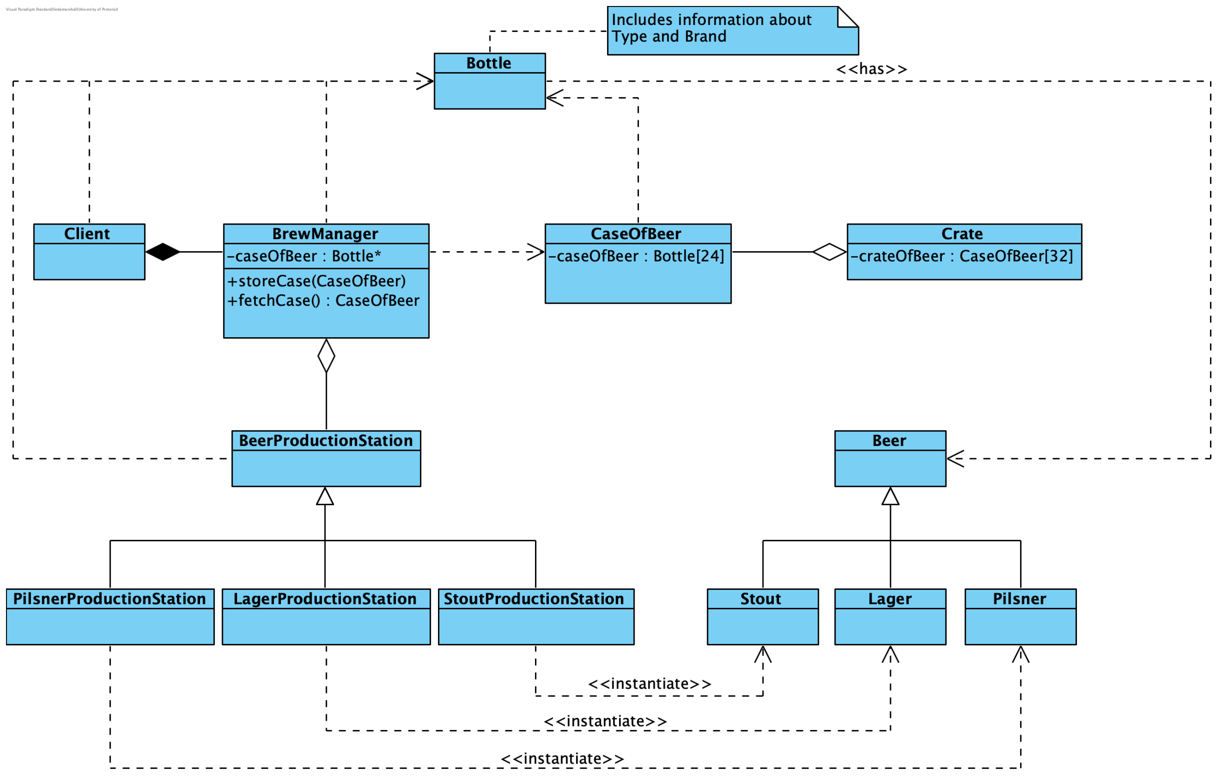
Solution: No, there is no wrapper for a case which would then be considered the Originator of the Memento pattern. Neither is there a Memento participant or a Caretaker participant.

An overview of the Memento class is given below. The Manager is the Originator, CaseOfBeer the Memento and Crate the Caretaker.



Part III

After visiting the local private brewery and sitting through an hour tutorial on how their processes work, you quickly sketch out the following high level design for your design team and programming team to review.



Question 7 ....(9 marks)

7.1 Which class represents the Memento participant? (1)

Solution: CaseOfBeer

7.2 From the information presented in the class diagram, is it possible to say that a Template Method (1) design pattern is to be implemented?

Solution: No

7.3 An extract from the class definition of `BeerProductionStation` is given below: (3)

```
class BeerProductionStation {

public:
  BeerProductionStation();
  void setupBrewingStation(int);

private:
  virtual void setupType();
  virtual void setBrands(int);
};
```

a) Which function would represent the template method? (2)

Solution: setupBrewingStation()

b) Which functions need to be implemented in the derived classes? (2)

Solution: setupType() and setBrands()

Question 8 .... (21 marks)

Consider the following definition of the `BrewManager` class and answer the questions that follow.

```
class BrewManager {

public:
BrewManager();
void setupBrewingStation(int);
bool brewBeer(string);
void cleanBrewingStation();
vector<Bottle*> shipCrate();

protected:
bool storeCase(Bottle*  CaseOfBeer);

private:
Bottle* caseOfBeer;
BeerProductionStation* beerProductionStation;
Crate* crate;
};
```

8.1 Which operations represent the `createMemento` and `setMemento` operations of the Memento pattern? You may assume that there are getters and setters defined in the `CaseOfBeer` class. Note, a handle to memento is not held by the Originator in this case. The Originator, creates the memento, populates it and then places it into storage. When the stored mementos are required (either a full or partially filled crate), the Originator will fetch the crate and pass it back to its client. (2)

Solution:

- `createMemento` - **bool** storeCase(Bottle∗ CaseOfBeer);

- `setMemento` - vector<Bottle∗> BrewManager::shipCrate();

8.2 The `setupBrewingStation` operation determines which ConcreteCreator is used. `brewBeer` calls the appropriate `produce` operation. Write the code for `brewBeer`. (3)

Solution:

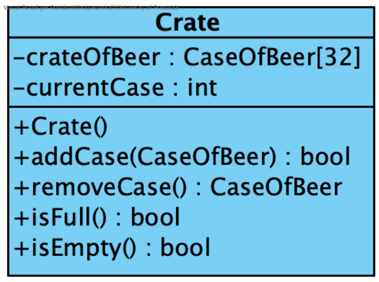
```
bool BrewManager::brewBeer(string brand) {
caseOfBeer = beerProductionStation->produce(brand);
return storeCase(caseOfBeer);
}

```

8.3 Why are the operations of the Memento participant defined as protected? (2)

Solution: This is so that all classes other than the Originator participant have a narrow interface to the Memento participant, that is `CaseOfBeer`.

8.4 The UML class diagram for the `Crate` class is given by: (14)


Crate
-crateOfBeer : CaseOfBeer[32]
-currentCase : int
+Crate()
+addCase(CaseOfBeer) : bool
+removeCase() : CaseOfBeer
+isFull() : bool
+isEmpty() : bool

- The constructor initialises `currentCase` to indicate that the crate is empty on construction.
- If the crate is not full, a case of beer is inserted into the crate and `true` is returned.
- A case of beer is removed from the crate if the crate is not empty and the case is returned. A place holder is returned if the crate is empty.

Assume that the class definition is given in the file `Crate.h`, provide the implementations that will be placed in the file `Crate.cpp`.

Solution:

```
#include "Crate.h"  // 1 mark

Crate::Crate() {. // 1 mark for the constructor
currentCase = -1;
}

bool Crate::addCase(CaseOfBeer caseOfBeer) { // 4 marks for addCase
if (isFull())
return false;
else {
currentCase++;
crateOfBeer[currentCase] = caseOfBeer;
return true;
}
```

```
    }

    CaseOfBeer Crate::removeCase() { // 4 marks for removeCase
    if (currentCase == -1)
    return EmptyCaseOfBeer();
    else {
    currentCase--;
    return crateOfBeer[currentCase+1];
    }
    }

    bool Crate::isFull() { // 2 marks for isFull
    return currentCase == 31 ? true : false;
    }

    bool Crate::isEmpty() { // 2 marks for isEmpty
    return currentCase == -1 ? true : false;
    }


```

Question 9 ....(3 marks)

The `Bottle` class is defined as follows:

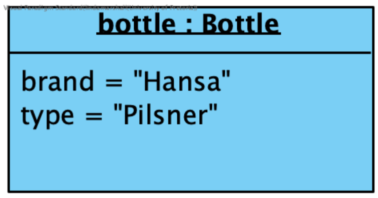
```
class Bottle {
public:
Bottle();
void setBrand(string);
void setType(string);
string getLabel();
string getCap();
private:
string brand;
string type;
};
```

For each bottle of beer, the type and brand of the beer contained in the bottle needs to be set. Assume an object of Bottle has been created as follows:

```
Bottle* bottle = new Bottle();
bottle->setBrand("Hansa");
bottle->setType("Pilsner");
```

The `getLabel` operation always includes both the brand and the type while the `getCap` operation only displays the brand. Draw the object diagram immediately after these three statements have executed.

Solution:



Question 10 .... (12 marks)

Consider the following incomplete main program and answer the questions that follow:

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
BrewManager* brewManager = new BrewManager();
int choice;
string brand;
string type;
bool crateFull;

do {
cout << "Which type of Beer would you like to brew?" << endl;
cout << "1. Pilsner" << endl;
cout << "2. Lager" << endl;
cout << "3. Stout" << endl;
cout << "4. I'm done brewing" << endl;

cin >> choice;

if ((choice < 4) && (choice > 0)) {
cout << "What is the brand? - ";
cin >> brand;

// include code to set up a brewing station
//   brew the beer and clean the brewing station here

}

} while ((choice != 4) && (!crateFull));

cout << "Either stopped brewing or crate is full" << endl;

// Check the crate
vector<Bottle*> crate =  // get the crate of beer, whether full or not.

for (std::vector<Bottle*>::iterator it = crate.begin() ; it != crate.end(); ++it) {
cout << "Beers in crate " << *it << endl;
Bottle* box = *it;
cout << "Beers in box " << box << endl;
for (int i = 0; i < 24; i++) {
// print the label on the bottle.
}
cout << endl;
}


delete brewManager;

return 0;
}
```

10.1 Provide the code to include all necessary header files. (4)

Solution:

```
#include "BrewManager.h"
#include "Crate.h"
#include "CaseOfBeer.h"

```

10.2 Write the code to set up a brewing station, brew the beer and clean the brewing station. (6)

Solution:

```
brewManager->setupBrewingStation(choice);
bool crateFull = brewManager->brewBeer(brand);
brewManager->cleanBrewingStation();

```

10.3 Complete the statement to receive the crate of beer that is either partially full or completely full. (1)

Solution: brewManager−>shipCrate();

10.4 Write a statement that will display the label on the current bottle from the case. (1)

Solution: cout << "Beer␣−␣" << box[i].getLabel();