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

COS 214 Class Test 2

- This test takes place on **3rd September 2021**.
- The maximum duration of this test is **40 minutes**.
- This test consists of **4 questions** for a total of **40 marks**.

Question 1 ..... (4 marks)

For each of the following descriptions, provide the name of the design pattern that best suits the description.

1.1 A pattern that creates an object by making a copy of an existing object. (1)

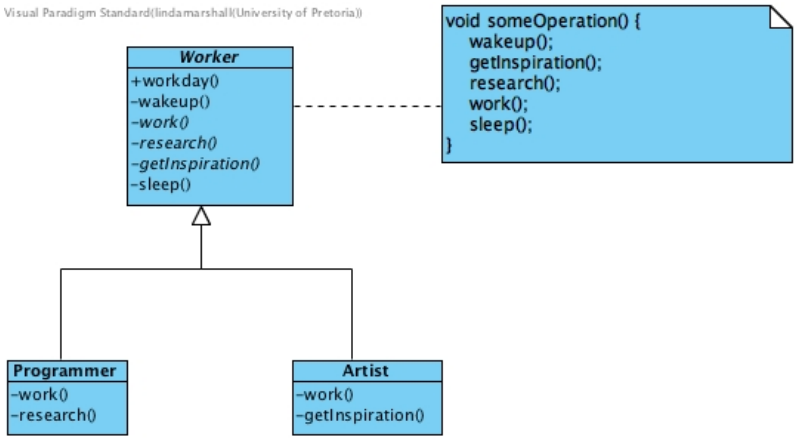
1.2 This pattern promotes rollback to full object status. (1)

1.3 A pattern that provides a hierarchy that encapsulates: many possible "platforms", and the construction of a suite of "products". (1)

1.4 This design pattern is used in situations where it is necessary to dynamically swap out algorithms. (1)

Question 2 ..... (18 marks)

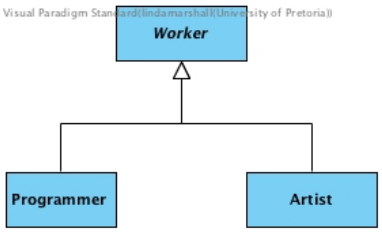
Consider the following class diagram and answer the questions which follow.



Assume you wish to create objects of `Worker` subclasses from a parallel hierarchy providing a uniform interface for creating `Worker` objects.

2.1 Which design pattern would you use to build the described functionality? (2)

2.2 Complete the following class diagram showing how you will add the required functionality to the given class diagram. You need not provide any details of the classes, class names will suffice. (7)



2.3 Name all the participants in the class diagram you completed in Question 2.2. Make sure you link them to the corresponding class in the diagram. (6)

2.4 Assuming you wish to add another worker type to your system. What will you need to define and implement? (3)

Question 3 ..... (10 marks)

There are a number of algorithms which generate pseudo-random numbers. Usually these algorithms are encapsulated in classes. Instantiations of these classes are then able to provide these pseudo-random numbers when queried. Usually these algorithms accept a seed, which is simply a user supplied number from which the first pseudo-random number is generated in a series of numbers. After this, subsequent numbers are generated as some function of the previous numbers in the sequence. The result of this is that two random number generators initialised with the same seed will generate an identical sequence of pseudo-random numbers.

Consider the `RNGenerator` class, which specifies an interface for generating random numbers but defers the actual calculations of these numbers to its subclasses, and the `Mersenne` class, which inherits from `RNGenerator`:

```
1  class RNGenerator {
2    public:
3      //Returns the next number in the sequence and
4      //generates its successor
5      virtual int getNextNumber()  {
6        int currentNumber = nextNumber;
7        setCurrentNumber(generateNumber(currentNumber));
8        return currentNumber;
9      }
10
```

```
11      virtual RNGenerator* clone() = 0;
12
13    protected:
14      //Constructor, only visible to subclasses
15      RNGenerator(int s)  {
16        seed = s;
17      }
18
19      //Copy Constructor
20      RNGenerator(const RNGenerator& other)  {
21        seed = other.seed;
22      }
23
24      //Assignment Operator
25      RNGenerator& operator=(const RNGenerator& other)  {
26        if(this == &other)
27          return *this;
28
29        seed = other.seed;
30        nextNumber = calculateInitial(seed);
31
32        return *this;
33      }
34
35      //Sets the current number in the sequence
36      virtual void setCurrentNumber(int s)  {
37        nextNumber = s;
38      }
39
40      //Returns the current number in the sequence without
41      //generating the next number
42      virtual int getCurrentNumber()  {
43        return nextNumber;
44      }
45
46
47      //Uses the seed to generate the first number in the sequence.
48      virtual int calculateInitial(int s) = 0;
49
50      //Returns the next random number in the sequence
51      virtual int generateNumber(int previousNumber) = 0;
52
53      virtual int getSeed()  {
54        return seed;
55      }
56
57    private:
58      int seed;
59      int nextNumber;
60  };
61
62  class Mersenne : public RNGenerator  {
63    public:
64      Mersenne(int s) : RNGenerator(s)  {
65        start = 0;
66        setCurrentNumber(calculateInitial(s));
67      }
```

```
68
69      Mersenne* clone()  {
70        return new Mersenne(*this);
71      }
72
73    protected:
74      //Implemented in the .cpp file
75      virtual int calculateInitial(int s);
76
77      //Implemented in the .cpp file
78      virtual int generateNumber(int previousNumber);
79
80      Mersenne(const Mersenne& other):RNGenerator(other)  {
81        this->start = other.start;
82      }
83
84    private:
85      int start;
86      static const int MAXSTART = 10;
87  };
```

3.1 Is the base class defined as abstract? (Yes/No) (1)

3.2 Identify the participants that match the given classes. (2)

1. RNGenerator:
2. Mersenne:

3.3 The `clone` function of the `Mersenne` class relies on the `protected` copy constructors. It creates a copy of a `Mersenne` object and both objects are initialised with the same seed, meaning that they will generate the same sequence of numbers. However, if the object of which the clone was created already generated random numbers, then the two `Mersenne` objects will not return the same numbers when queried in parallel. Reimplement the `clone` function such that the following code snippet will always output **true**: (3)

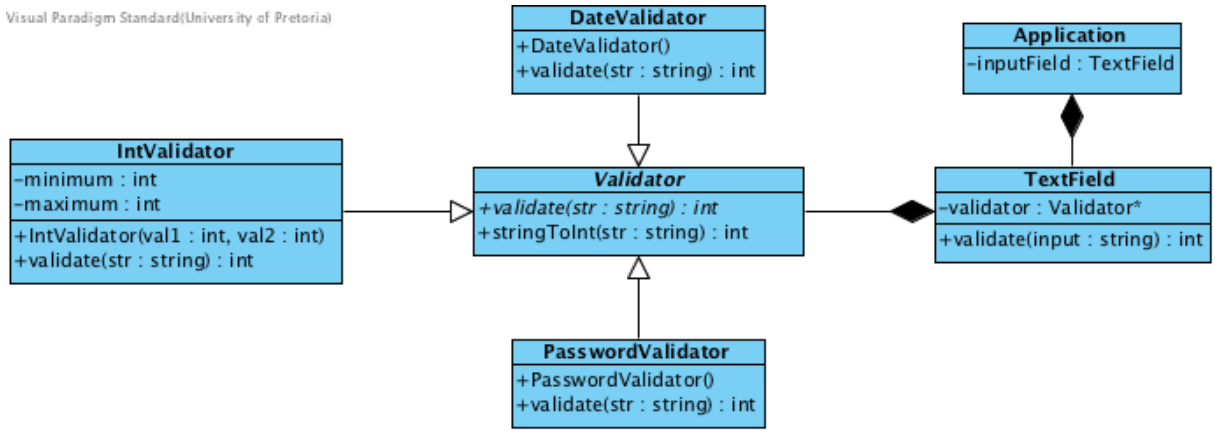
```
1  Mersenne* one = new Mersenne(6);
2
3  for(int i = 0; i < 5; ++i)
4    one->getNextNumber();
5
6  Mersenne* two = one->clone();
7  bool same = true;
8
9  for(int i = 0; i < 10; ++i)
10   same = one->getNextNumber() == two->getNextNumber();
11
12 if(same)
13   cout<<"true"<<endl;
14 else
15   cout<<"false"<<endl;
16
17 delete one;
18 delete two;
```

3.4 Draw a UML class diagram showing the relationship between the `RNGenerator` and `Mersenne` classes. Do not draw any of the class features for the classes involved in the relationship. (3)

3.5 What is the relationship between the code snippet given in Question 3.3 for the object created in Line 1 and the class `Mersenne`? (1)

Question 4 ..... (8 marks)

The following diagram is the class diagram of the design of a generic input field used by a client program. It applies the Strategy design pattern.



4.1 Identify the participants of the pattern in the given design. (3)

1. Context
2. A Concrete Strategy
3. Strategy

4.2 Which method maps onto the `algorithm()`? (1)

4.3 Write the implementation of the `validate()` method of the `TextField` class as it should appear in `TextField.cpp`. (4)