UEF 7.3. Design Patterns

Tutorial 2

Creational patterns

Exercise 1 : (CI 2017/2018)

Let the code given below :

1. Give the class diagram corresponding to this code.
2. What design pattern is used for the creation of animals ?
3. Why did we use the “instances” attribute ?
4. Suggest a new solution without this attribute and using a design pattern.
5. Give the new code.

```
public abstract class AnimalU {
 public static Map<String,Animal> instances= new HashMap<String, Animal>();
 //…
 public abstract Animal instantitateAnimal();
}

public interface Animal{
// Methods
}
public class Cat implements Animal {
// Methods
}
public class Dog implements Animal {
// Methods
}
public class CatU extends AnimalU{
 public Animal instantitateAnimal() {
 if ( instances.containsKey ( "Cat" )) {
     return (Animal) instances.get( "Cat" );
 }
 else {
   Cat c= new Cat();
   AnimalU.instances.put("Cat",c) ;
   return c ;
   }
 }
}
public class DogU extends AnimalU{
 public Animal instantitateAnimal(){
  if ( instances.containsKey("Dog")) {
   return (Animal)instances.get("Dog" ) ;
  }
  else {
    Dog c= new Dog();
    AnimalU.instances.put("Dog",c);
    return c ;}
  }
 }
}
```

Exercise 2 (CI 2019/2020) :

We want to write a generator of SQL queries (relational database), OQL (relational database object) or even SPARQL queries (RDF graphs)... For this, we need to build queries of different syntaxes according to the language but which use (for simplifying) the same input data: the fields that we want to retrieve (SELECT), the data sources(FROM) and the selection condition(WHERE).

For example, to generate the following SQL request:

```
SELECT person.id, person.name FROM person;
```

the following code is used:

```
QueryGenerator req = new SQLQueryGenerator();
req.select( "person.id" );
req.select( "person.name" );
req.from( "person" );
Request sql=req.getResult();
```

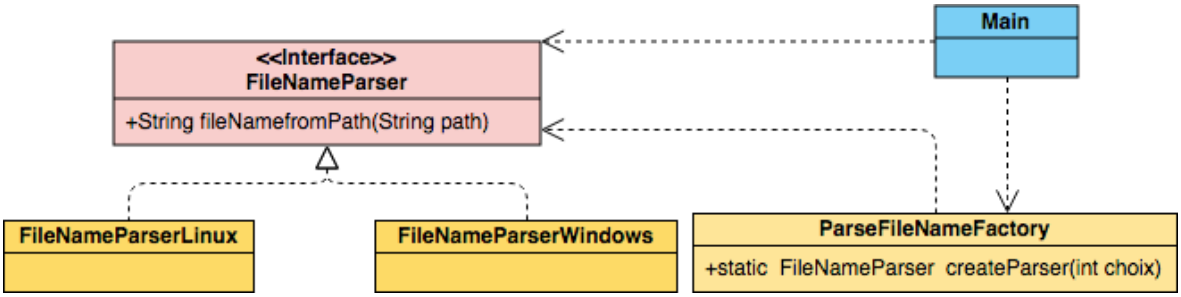
1. Give the class diagram allowing to have this processing of the request using a GoF pattern. Explain.
2. We wish that each request language generator be unique.
   a. What is the better adapted GoF pattern that can answer this requirement ?
   b. Give the code parts allowing to ensure this solution the SQL request generator.

Exercise 3 (adapted from CI 2019/2020) :

Let the program Main below, that allow to extract and display the file name from a given Windows path (If you test this program by giving " C:\Windows\hello.dll ", it should display " hello.dll ").

```
public class Main{
 public static void main(String [] args){
 String path=args[0] ;
 // index is the last position in path of the "\" character
 int index = path.lastIndexOf("\\");
 //We built a string that contains only parts located right of the index
 String r =path.substring(index+1);
 System.out.println(r);
 }
 }
```

We want to evolve this program by adding the possibility of extracting and displaying a file name from a given Linux file path (for example in "/user/share/hello.rc", the file name is hello.rc). Thus, we propose the following solution:



1. **a.** Does this solution implement a GoF design pattern? Explain (if yes which one and give the correspondences with the elements of the pattern. If not, explain Why).
   b. Give the new Java code of the class Main allowing to create a Windows Or Linux FileNameParser according to the user choice.
   c. Change the code to apply the Factory method pattern.

By maintaining this solution, we would like to be able to get also from this program (the class Main) the number of folders that must be browsed from the root to find our file. For this, we suggest adding a CountFolders interface, and two sublasses that will implement it (one for Windows, and one for Linux).

2. **a.** What GoF pattern will allow us to guarantee that the implementation of the

instantiated CountFolders will correspond to the FileNameParser depending on the operating system chosen by the user ? Explain

b. Give the corresponding class diagram, explaining the correspondences with the pattern.

c. Give the new Java code that implement your solution

Exercise 4 : Plane Action Hero – Creational Patterns in Action

“Plane Action Hero” is a 2D arcade game that simulates the progress of a plane that continuously firing bullets or missiles, collecting bonuses, avoiding traps and battling enemies.

A plane is characterized by a state (from 0 to 100, if it reaches 0, the player loses), a speed, and a striking power.

The plane advances through several levels, facing different enemies, collecting bonuses, and avoiding traps.

The game needs a central component “GameEngine” to keep track of the current score and level, manage the player’s plane, handle creation of enemies and bonuses. At any given time, only one instance of this component should exist.

A/

1. Why would it be problematic if multiple “Game Engines” existed simultaneously?
2. How could you ensure that only one instance of this class can ever exist?
3. How could this class be globally accessible while keeping it encapsulated?

The player faces the following enemies : Helicopters, FighterPlane, BossPlane. Each enemy fighter plane is composed of the following elements :

- A carcass of which the speed varied from 1 to 20
- A spear missile of which the strength is of 1 to 10
- A shield of which the strength varied from 0 to 10

B/

1. How can you delegate the responsibility of creating enemies so that new ones can be added without modifying existing code?
2. Some enemies share the same structure but differ in how their components are assembled. Could you separate what is built from how it is built?
3. Design a flexible way to create different enemies without changing the client code that uses them.

The game is divided into several levels. Each level defines its own set of enemies, bonuses, and traps.

Level	Enemies	Bonuses	Traps
Level 1	Helicopters	Shield Booster, Missile	Bomb
Level 2	Fighter Planes	Random Bonus, Cloner	Slowdown
Level 3	Boss Planes	Random Bonus	Poison

C/

1. How can the game engine easily switch between level configurations (e.g., Level 1 vs Level 2)?
2. How can you ensure that all objects created within a level are consistent with its theme or difficulty?
3. If a new level is added, how can it be integrated without modifying the code for the previous ones?
4. Create a flexible way to define and instantiate level-specific families of enemies, bonuses, and traps.

Some bonuses have special effects, such as:

- Cloner : creates a temporary duplicate of the player's plane.
- Shield Booster : adds temporary shield protection.

D/

1. How can you duplicate complex objects (like a plane with multiple attributes) without knowing their class type?
2. Implement an elegant way to duplicate or replicate game entities when needed.

Now that you have designed your creation mechanisms:

- integrate them into the GameEngine,
- make the engine capable of starting any level and generating the appropriate elements.

E/

1. How do all your creation mechanisms interact?
2. Which part of your code should know *how* objects are created, and which part should simply *use* them?
3. Draw an UML class diagram of your final design.