Factory Pattern

- هي طريقة بفصل فيها ال creation بتاع ال objects عن باقي ال system
objects المقصودة هنا بتكون معمولوها encapsulate وبيتم التعامل معاها من خلال interface
فباقي ال system مش عارف ايه ال concrete object اللي محتاج اعمله creation ولكنه يطلب
من ال factory يعمله interface object واللي بيحدد ال concrete object هو ال factory.

- <u>طريقة القيام بهذا من خلال</u>
  1- عمل Factory class مسئول عن creation بتاع ال concrete objects.
  2- ال client class بيتعامل مع ال Objects من خلال ال interface بغض النظر عن نوعها.
  3- Factory class بياخد من ال client طلب ال creation لل object وبياخد من ال configuration class او ال GUI مواصفات ال object اللى بتعبر عن ال concrete object.

<u>مثال:</u>
عندى shape interface class
وال concrete classes هي ال circles وال rectangular
ال client جواه pointer to shape وبيطلب من ال factory اعملي shape بياخد من ال configuration نوع ال shape فيعمل circle object او rectangular object
وبعدين هيرجع ال object لل client فيعمل draw لل shape بغض النظر عن نوعه.

- <u>مميزات ال Factory Pattern</u>
  1 – ال client بقي بيستخدم shape ومش هو اللي بيعمله creation (strong cohesion)
  2 – ال client ميعرفش ال shape ده حقيقتاً بيعبر عن أي concrete object هو عارف بس ان ده shape بالتالى لو حبيت اضيف concrete object جديد مش محتاج اعدل فيه (loose coupling).
  3 – خليت ال system اكثر سهولة في تقسم الشغل مثلا على ال developers فاخلى واحد ينفذ ال factory class والأخر ال client والاتنين مش هيعتمدوا على بعض نفس الفكرة لو بعمل testing هقسم ال testing كل class لوحدة من غير أي dependence.

```cpp
#include <string>

using namespace std;

//interface class
class Connection
{
public:
	Connection() {};
	virtual string description() { return "Generic"; };
};

//concrete product of factory
class OracleConnection : public Connection
{
public:
	OracleConnection() {};
	string description() { return "Oracle"; };
};

//concrete product of factory
class SqlServerConnection : public Connection
{
public:
	SqlServerConnection() {};
	string description() { return "SqlServer"; };
};

//concrete product of factory
class MySqlConnection : public Connection
{
public:
	MySqlConnection() {};
	string description() { return "MySql"; };
};
```

```cpp
#include <iostream>

//configuration
class configuration
{
	string _ConnectionType;
public:
	configuration() {};
	void setConnection()
	{
		cout << "Insert Type of connection" << endl;
		cin >> _ConnectionType;
	}
	string readConnection() { return _ConnectionType; };
};
```

```
#include <algorithm>

//factory
class Factory
{
	string _type;

public:
	Factory();
	Connection* createConnection();
};

Factory::Factory()
{
	configuration conf;
	conf.setConnection();
	_type = conf.readConnection();
}

Connection* Factory::createConnection()
{
	string type = _type;
	transform(type.begin(), type.end(), type.begin(), ::tolower);

	if (type == "oracle")
	{
		Connection* oracle = new OracleConnection();
		return (oracle);
	}
	else if (type == "sqlserver")
	{
		Connection* sqlServer = new SqlServerConnection();
		return (sqlServer);
	}
	else if (type == "mysql")
	{
		Connection* mySql = new MySqlConnection();
		return (mySql);
	}
	else
	{
		return NULL;
	}
}
```

```
//client
void main()
{
	Factory f;
	Connection* conn = f.createConnection();

	if (conn == NULL)
		cout << "unavilable connection" << endl;
	else
		cout << "You're connecting with " << conn->description() << endl;
}
```