ADDIS ABABA UNIVERSITY (AAU)

ADDIS ABABA INSTITUTE OF TECHNOLOGY (AAiT)

INFORMATION TECHNOLOFY AND SCEINTIFIC COMPUTING (ITSC)

SOFTWARE ENGINEERING

EXPLANATION FOR CHOOSEN DESIGN PATTERN

GROUP MEMBERS

NAME	ID NUMBER
1. KENA WAKWOYA	ATR/4296/08
2. ETSEGENET MELESE	ATR/8845/08
3. MILKIAS TONJI	ATR/8137/08

SUBMITTED TO MR. ALAZAR ALEMAYEHU

Q1. Singleton design pattern

Why do we choose singleton design pattern solution?

What is the singleton design pattern?

The singleton pattern is one of the simplest design patterns: it involves only one class which is responsible to instantiate itself, to make sure it creates not more than one instance; in the same time it provides a global point of access to that instance. In this case the same instance can be used from everywhere, being impossible to invoke directly the constructor each time.

Importance of the singleton pattern?

- ✓ There only one class is needed to instantiate an object
- ✓ Provide a global point of access to the object
- ✓ Are used for centralized management of internal or external resource
- ✓ A robust singleton implementation should work in any conditions
- ✓ It works when multiple threads uses it.

So, we did not create multiple instance of an object with in the class. And it is very important to use singleton pattern when dealing with database classes. Once we create an instance of the object we can use repeatedly.

The Singleton pattern is useful when you want to reuse some expensive to create resource. Since the creation is performed only once for the entire lifetime of the application, you are paying the price only once.

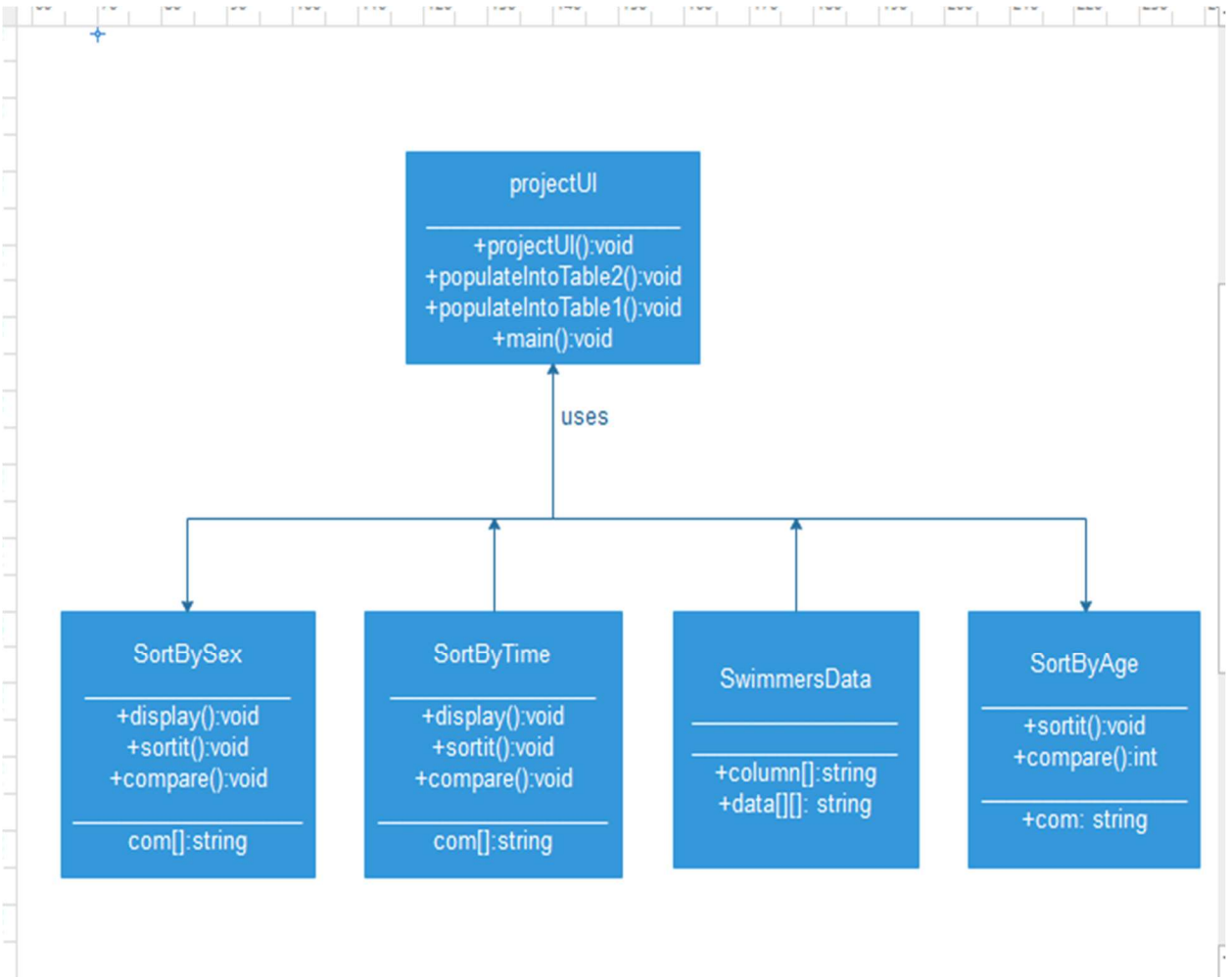
Singleton pattern is useful when a resource is expensive in terms of occupied memory or time that it takes to be created. Another usage that can be found to the singleton pattern is that you can restrict the number of instances of a class base on you application requirements, one example that comes in mind is the connection to a database, where you will always make one connection per thread, that connection being usually alive along with thread.

Singleton is essentially a concept stating that some entity must have only single instance. The reasons for single instances can vary, these are the main ones:

- ✓ Reuse a shareable resource in different modules.
- ✓ Disallow multiple instances of a resource.

- ✓ Create an instance when any using module isn't ready yet - e.g., a global variable that is created and initialized before any program code runs, so when the using modules are created, the global is ready.

Class diagram of singleton design pattern



Q2.Bridge design pattern

Why do we choose bridge design pattern solution?

What is bridge design pattern?

Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete class independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

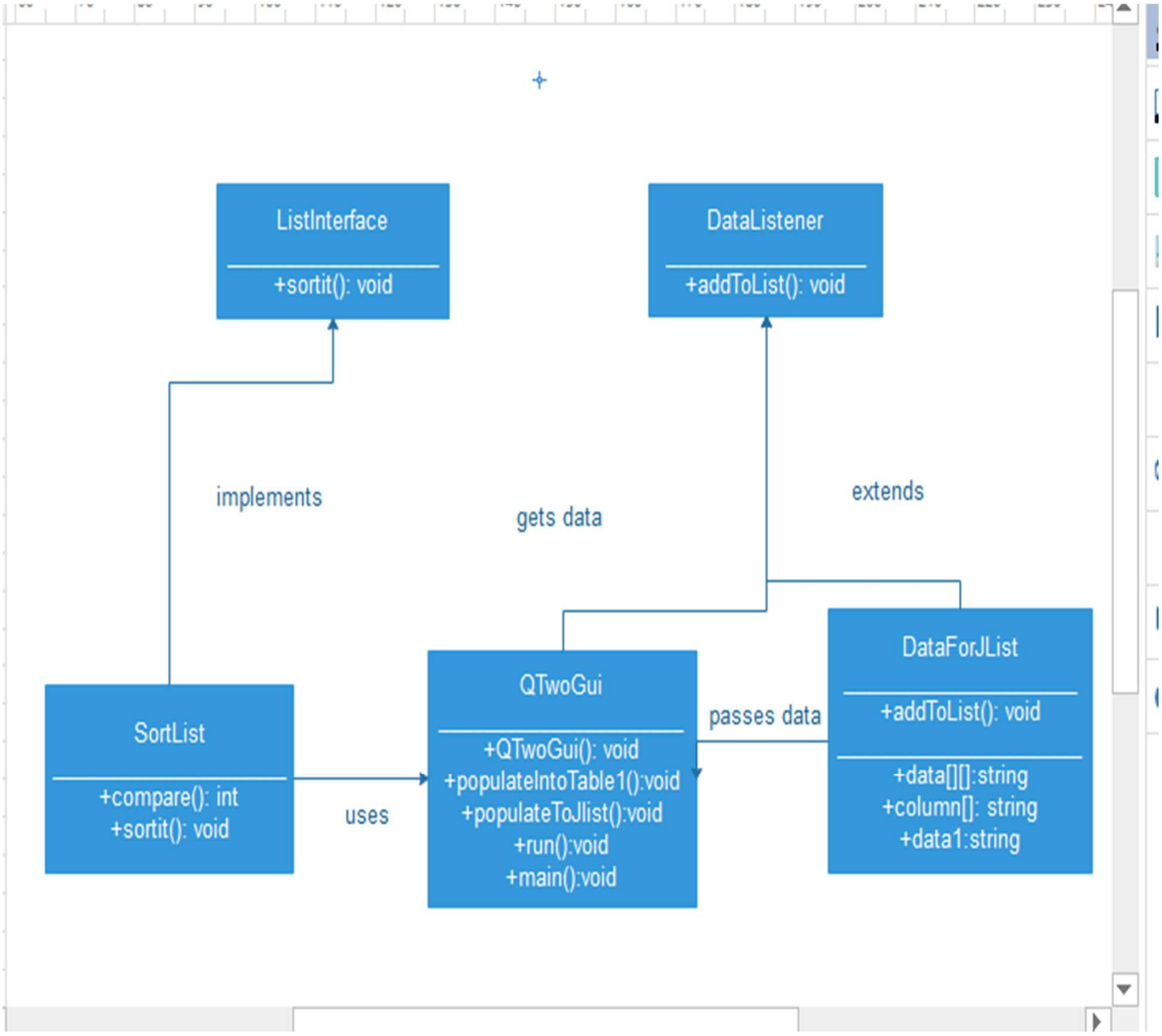
Importance of bridge design pattern

- ✓ Separation between abstraction and implementation details
- ✓ Promote independent evolution of the implementations
- ✓ used for code that is more likely to be greenfield
- ✓ defining the interface of implementing classes
- ✓ used to run-time bind implementation
- ✓ share an implementation among multiple objects

We have four components such as abstraction, concrete classes, interfaces and redefined abstractions. So, to bridge the connection between these all components we found the best suitable pattern is as bridge design pattern.

- ✓ Abstraction: it defines the interface
- ✓ Implementer: it defines an interface for the implementation
- ✓ Concrete classes: implements the interface
- ✓ Redefined implementer: it implements the abstraction

Class diagram of the bridge design pattern that we have used



Q4.Composite pattern design

Why do we choose composite design pattern solution?

What is the composite design patter?

The **composite pattern** is a partitioning design pattern. The composite pattern describes a group of objects that is treated the same way as a single instance of the same type of object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly

It describes how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse.

Importance of composite design pattern?

- ✓ A part-whole hierarchy should be represented so that clients can treat part and whole objects uniformly.
- ✓ A part-whole hierarchy should be represented as tree structure.
- ✓ Define a unified `Component` interface for both part(Leaf) objects and whole (`Composite`) objects
- ✓ Individual `Leaf` objects implement the `Component` interface directly, and `Composite` objects forward requests to their child components.
- ✓ using multiple objects in the same way, and often have nearly identical code to handle each of them
- ✓ It is less complex in this situation to treat primitives and composites as homogeneous.
- ✓ Recursive composition
- ✓ "Directories contain entries, each of which could be a directory."
- ✓ 1-to-many "has a" up the "is a" hierarchy

Class diagram of the composite design pattern that we have used