nterface and Abstract Class

1. example

Graphical user interface, application

Description automatically generated

Fish can be animal as well, but in JAVA because of JVM binding principle, a class can does not have multiple parents.

Change food to interface, then fish can inheritate from animal class.

Graphical user interface

Description automatically generated

1. Features

* All the attributes are default to public, static, final, and must be initialized explicitly. Can not define other type.
* All the methods are default to public, abstract. Can not define other type.
* Interface do not have constructor, can not declare instance of interface.
* One interface can not implement another interface but can inheritate multiple interfaces.
* Class implement interface. Syntax as below

public class Glass implements Transparency{…}

* All the methods in interface must be implemented in class, or else the class should be declared as abstract class.
* A class can only have one parents but can extends multiple interfaces.

1. Difference between Abstract class

* All at top of family tree.
* Can not declare instance.
* In Abstract class, can define implementation for part of methods, but in interface, you can not do this.
* Any modification of implemented methods has no impact of his child. But in interface, if you add a method, all the class implement this interface got to change to implement the newly added method.

1. Accessing interface (same as abstract class)

Define interface

Define class to implement interface

In client class, declare [interface name] myInterface = new ClassImpl()

Java design pattern

1. Factory Method Patten

Its about using abstract class/interface. Define an interface or abstract class for creating an object but let a factory subclass to decide which class to instantiate.

Diagram

Description automatically generated

Related Knowledeg:

* Usage of abstract class
* Usage of java.util.properties, format of properties file and how to load
* Usage of getResourceAsStream to load file into InputStream
* Java.lang.class.forName(className).newInstance.

Advantage of Factory Pattern:

* High maintenance performance, if we want to add one more shape triangle, just add one more item in properties file, don’t have to change existing code at all.

Code example:



1. Adapter Pattern

Structural design pattern converts the interface of a class to another interface the client expect.

UML example:

Two implementations of Adapter Pattern

* 1. Case1

Diagram

Description automatically generated

* 1. Case2

Diagram

Description automatically generated

Core: Adapter class need to implement the interface what client class suppose to call. Then override the method to call the method which client exactly want with either extend interface Usb2 implement class MyUsb2 or Aggregate class MyUsb2 into Adapter class.

Code example: refer to code folder

* 1. Case3

In the scenario of an interface defined lots of methods, but we only want to use one or a few of them. If we define a class to implement this interface, we got to implement all the methods. So, the method we do not use will be written as empty method. This is little bit messy in code. In this case, we use an abstract class as adaptor, this abstract class implement the interface, all the method in it will be empty, then we create a class to extend this abstract class, now we just need to implement the method we use.

Code snapshot

Interface a:

public interface A{

void a();

void b();

void c();

void d();

void e();

void f();

}

Adapter class

public abstract class Adapter implements A{

public void a(){};

public void b(){};

public void c(){};

public void d(){};

public void e(){};

public void f(){};

}

Implement class

Public class Impl extends Adapter{

public void a(){

System.out.println(“A is implemented”);

}

}

1. Proxy Pattern

Proxy pattern means an object representing another object.

It provides the protection to the original object from the outside world. Decuple from the original object, any change to the original object wont affect client. E.g use service proxy class to access DAO(database operation)

Example

UML

Diagram

Description automatically generated

Another example of java web design

1. MVC Pattern(not very clear)

Graphical user interface, diagram

Description automatically generated

A picture containing graphical user interface

Description automatically generated

1. Singleton Pattern

Define a class that has only one instance and provides a global point to access of it.(hint: no public constructor)

Two forms a singleton design pattern

* Early Instantiation: creation of instance at load time
* Lazy Instantiation: creation of instance when required.

Advantage of singleton design pattern:

* Save memory. Only single instance is reused again and again.

Usage of singleton design pattern

* Multi-thread, database application, logging, caching, thread pools, configuration settings.

Element of Singleton Pattern

* Static member
* Private constructor
* Static factory method: getInstrance()

1. Prototype pattern

Cloning of an existing object instead of creating new on and can also be customized as per the requirement. Like magic monkey use his hair clone himself.

Get a new instance from clone concrete class instead using new.

* Reduce the need of sub-classing
* Hides complexities of creating new objects
* Clients can get new objects without knowing which type of object it will be
* Lets you add or remove objects at runtime

General Realization（For all OO language）

Diagram

Description automatically generated

Java realization using Java.lang.object clone() method

Diagram

Description automatically generated

We can treat concrete class as a template, use this template can create lots object, then modify the object based on the attributes template provided.

Imagine our timesheet system, allow you save a template, so you don’t have to fill all the fields everytime, just populate timesheet with template then modify based on it. The template can be treated as concrete class.

Complicated case：

Concrete class contains reference to other object, so when clone this concrete class, they are two types clone

1>shallow clone: cloned instance will share the memory with concrete instance for the reference object.

2>deep clone: cloned instance is totally independent from concrete instance(template)

Diagram

Description automatically generated

Shallow clone realization: as of above

Deep clone realization: serializable.