Exercise 1

You need to create a class to manage preferences. In order to maintain consistency, there should only ever be one instance of this class.

- In order to maintain consistency, there should only ever be one instance of this class. Why ? What could happen ?
- How can you ensure that only one instance of a class is instantiated?
- **Write the Java code** of the class.

```
public class PrefManager {

    private PrefManager instance;
    private Map<String,Boolean> prefList;

    private PrefManager() {}

    public static synchronized PrefManager getInstance() {
        if(instance == null)
            ~~instance = new PrefManager();~~
            ~~prefList = new HashMap<String,String>();~~
            ~~prefList.put("fullscreen", false);~~
            ~~prefList.put("sound", true);~~
            ~~prefList.put("cheats", false);~~
        return instance;
    }

    public boolean getPreferenceForKey(String key) {
        return prefList.get(key);
    }

    public void addPreference(String key, boolean val) {
        prefList.put(key,val);
    }
}
```

That belongs in the constructor

```java
public class PrefManager {

    private PrefManager instance;
    private Map<String,Boolean> prefList;

    private PrefManager() {
        prefList = new HashMap<String,String>();
        prefList.put("fullscreen", false);
        prefList.put("sound", true);
        prefList.put("cheats", false);
    }

    public static synchronized PrefManager getInstance() {
        if(instance == null)
            instance = new PrefManager();
        return instance;
    }

    public boolean getPreferenceForKey(String key) {
        return prefList.get(key);
    }

    public void addPreference(String key, boolean val) {
        prefList.put(key,val);
    }
}
```

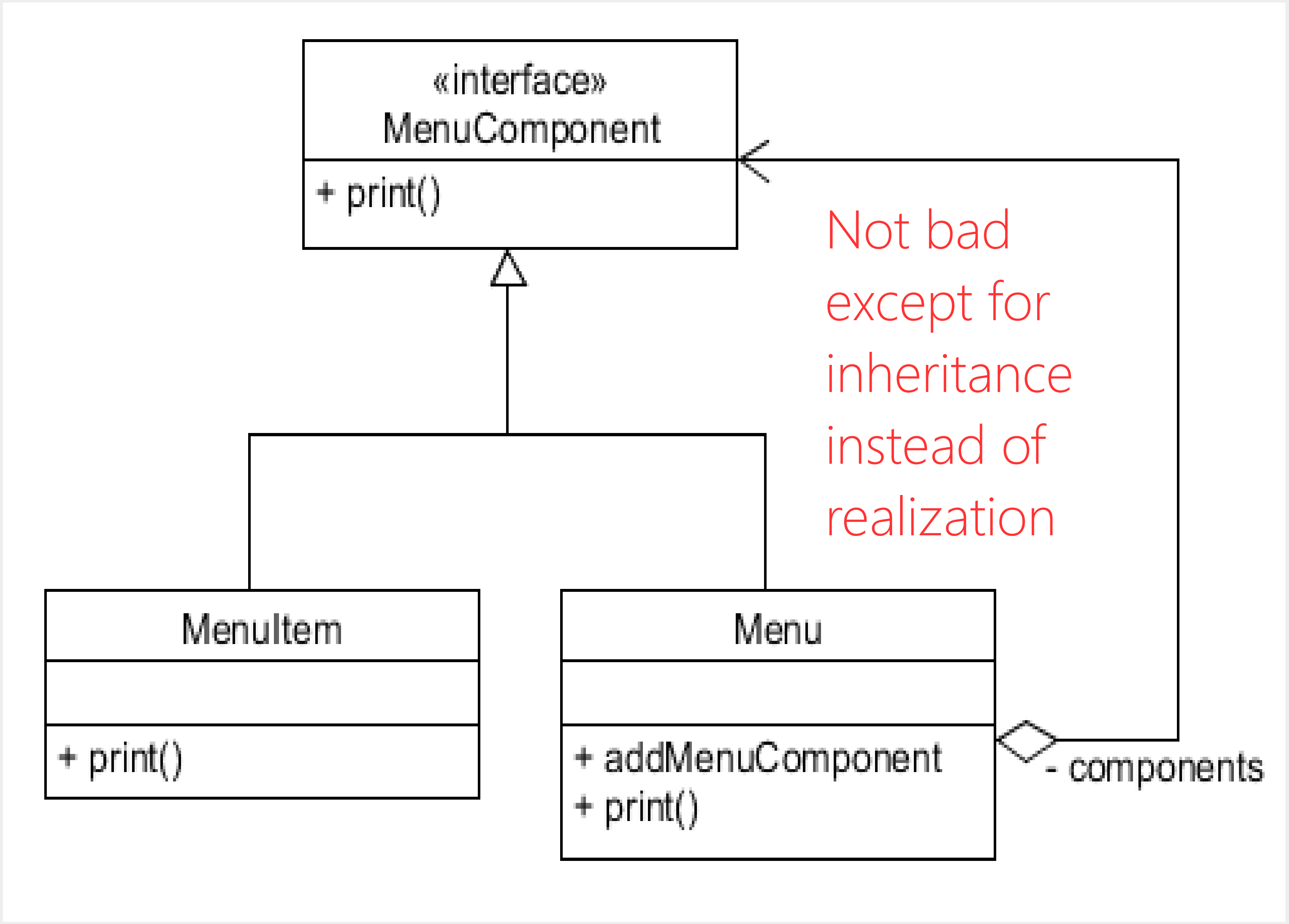
GOOD

Exercise 2

You are implementing a menu that has a recursive structure for a restaurant. Their menu contains (sub)menus and/or menu items. Each (sub)menu has (sub)menus and/or menu items.

You want to be able to represent this hierarchy, and you want to be able to easily perform operations on the whole menu, or any of its parts.

Draw a UML class diagram for the system



Not bad except for inheritance instead of realization

Exercise 3

Draw a UML class diagram for the system on the following slide for ordering Pizza.

Once you're finished drawing the UML diagram, **write the Java code** to implement the system.

Exercise 3

```
1. interface PizzaOrder {

boolean isVegan(); //true if it has no cheese or meat

boolean hasMeat();

double price(); //dollars

} //Don’t need a hasCheesemethod…
```

ArgoUML

2. BasicMediumPizza

– A regular pizza, nothing special, costs $10.00

– This pizza includes cheese by default (in real life), no need to additionally specify cheese (in the code)

3. NoCheese:Modifies a pizza so no cheese is used at no additional charge

4. Pepperoni : Modifies a pizza so pepperoni is added, for $1.00

5. Client : Creates a basic medium pizza, no cheese, with pepperoni. Checks the price and checks if it is vegan.