## Task 2

1. **Describe the principle of polymorphism and how and where it is being used in Task 1.**

Polymorphism, meaning "many forms," represents when different classes related by inheritance use methods to carry out different tasks. This enables a single action to be performed in many different ways according to the specialised methods in derived classes.

In task 1, polymorphism is used to treat both Batch objects and Transaction objects as Thing in the Sales class.

The “**Print**” and “**Total**” methods in the Thing, Batch, and Transaction classes are declared as abstract in the Thing class and overridden in the subclasses Batch and Transaction. This enables each subclass to offer its own implementation of these methods, crucial for printing and calculating totals tailored to each type of object.

The “**PrintOrders”** method in the Sales class iterates through a list of Thing objects and invokes their Print and Total methods. As Batch and Transaction are subclasses of Thing, they can be stored in the same list, allowing their methods to be called polymorphically. When Sales use a Thing object, it could be either a Batch or a Transaction. Polymorphism enables the order to be printed out differently depending on whether we’re printing a Batch or a Transaction.

1. **What is wrong with the class name Thing? Suggest a better name and explain the reasoning behind your answer.**

The name “**Thing**” is too generic in this context. If the program has more abstract class, the name Thing might make it difficult to tell which feature this group of classes represents. For example, in the Shape Drawing program, Shape class is the abstract class. By looking at the name, people can understand what this class illustrates. Therefore, to reflect better the purpose of the class, I would recommend this abstract class to be named “**SalesItem**” or “**OrderItem**”. This change makes the code more understandable to the coder. For instance, Sales class would be written to have a list of “**OrderItem**” instead of having a list of “**Thing**”.

1. **What is abstraction and how and where it is being used in Task 1.**

Abstraction is the process of hiding complex details and showing only essential features to the external environment. An abstract class serves as a blueprint for its derived classes, providing a basic structure without specifying the implementation of its methods.

In Task 1, abstraction is implemented through the use of the Thing class. The Thing class represents sales items, hiding specific order details like Batch and Transaction, and providing common methods like Print and Total.

The Print and Total methods in Thing are abstract, so subclasses must specialise them. This hides the specifics, letting each subclass customise its behavior. The Sales class manages a list of Thing objects, hiding individual order details and allowing uniform interaction with different order types.

1. **Can you think of a scenario or system design that resembles Task 1? Look at the classes and their interaction in Task 1 and identify a real-world system or approach that uses a similar relationship.**

Task 1 is like a basic sales system, managing different types of orders, such as batches and single transactions. It is similar to an online store with:

* **Thing** is the listed products on the platform.
* **Batch** is the orders containing many products.
* **Transaction** is a single purchase of customers.
* The **Sales** class handles order processing, total calculation, and reporting in the system.