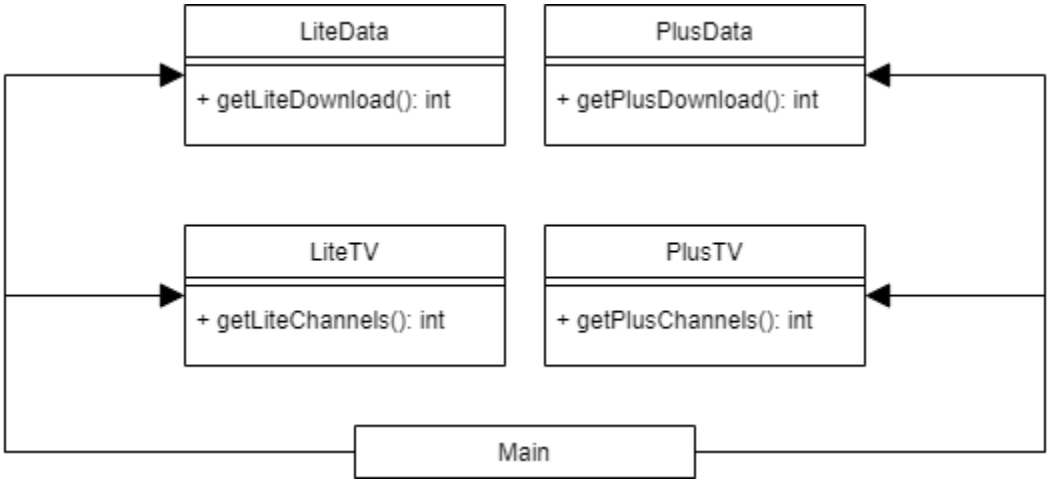
A cable service provider has an application that manages the packages it offers to customers. A package consists of a Data and TV bundle of the same type ie (Lite/Plus). The class diagram below outlines the functionality for representing the various bundles.
 or Data/TV



The Main class calls the bundle classes to create packages and have them print details about the package.

```
class Main {
  public static void main(String[] args) {
    String[] packages = {"Lite", "Plus", "Lite", "Lite", “Max”};
    for (String p: packages){
        if ( p == "Lite"){ //create a lite package
            LiteData ld = new LiteData();
            LiteTV lt = new LiteTV();
            System.out.println(
              p+" package has "+
              ld.getLiteDownload()+" download and "+
              lt.getLiteChannels()+" channels"
            );
        }else if (p == "Plus"){ //create a plus package
            PlusData pd = new PlusData();
            PlusTV pt = new PlusTV();
            System.out.println(
              p+" package has "+
              pd.getPlusDownload()+" download and "+
              pt.getPlusChannels()+ " channels"
            );
        }
    }
  }
}
```

The code above produces the following output:

```
Lite package has 100 download and 32 channels
```

```
Plus package has 250 download and 68 channels
Lite package has 100 download and 32 channels
Lite package has 100 download and 32 channels
```

Questions

1. By analyzing the UML diagram, identify any problem areas with the software design. **[3 Marks]**

   With the current approach; the client is directly dependent on several bundle classes and must be updated with the addition of new bundles

   Lack of abstraction, getLiteDownloads() getPlusDownloads() can be generalized to getDownloads() in a Data parent class but implemented differently. Similarly for getLiteChannels() and getPlusChannels() for a TV class.

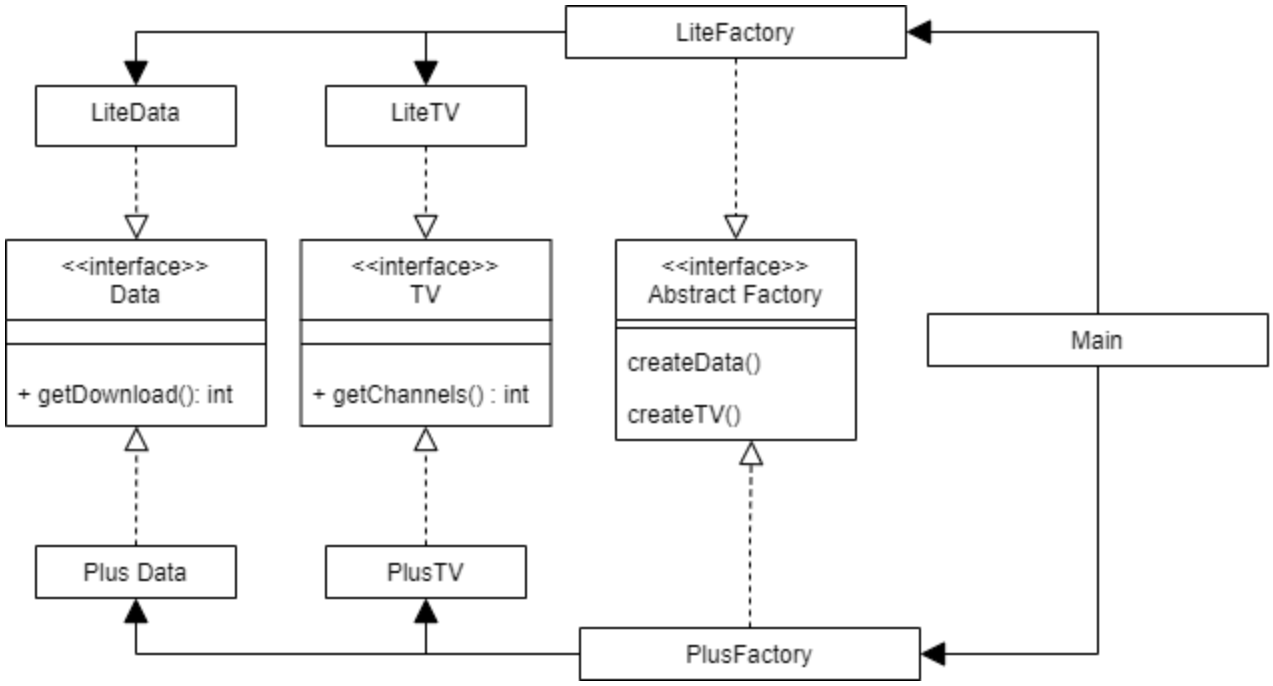
2. By analyzing the code provide criticisms for the implementation citing any solid violations **[4 Marks]**

   The client must be aware of the various bundles and the different methods they provide even though they are functionally the same. This can be argued as a dependency inversion violation as the client is tightly coupled to low level bundle classes but is mainly concerned with packages. The client must ensure that bundles of different types aren't mixed e.g. PlusData and LiteTV when creating packages.
   Would also accept SRP violations

3. Identify a creational design pattern (Singleton, Factory method, Abstract Factory, Builder) that addresses an issue in the current design and explain how. **[3 Marks]**

   The abstract factory method can be introduced to abstract the bundle classes from the user. The user can use the factory to create light or plus packages instead of instantiating individual bundle classes thus addressing the issue of too many dependencies in main.

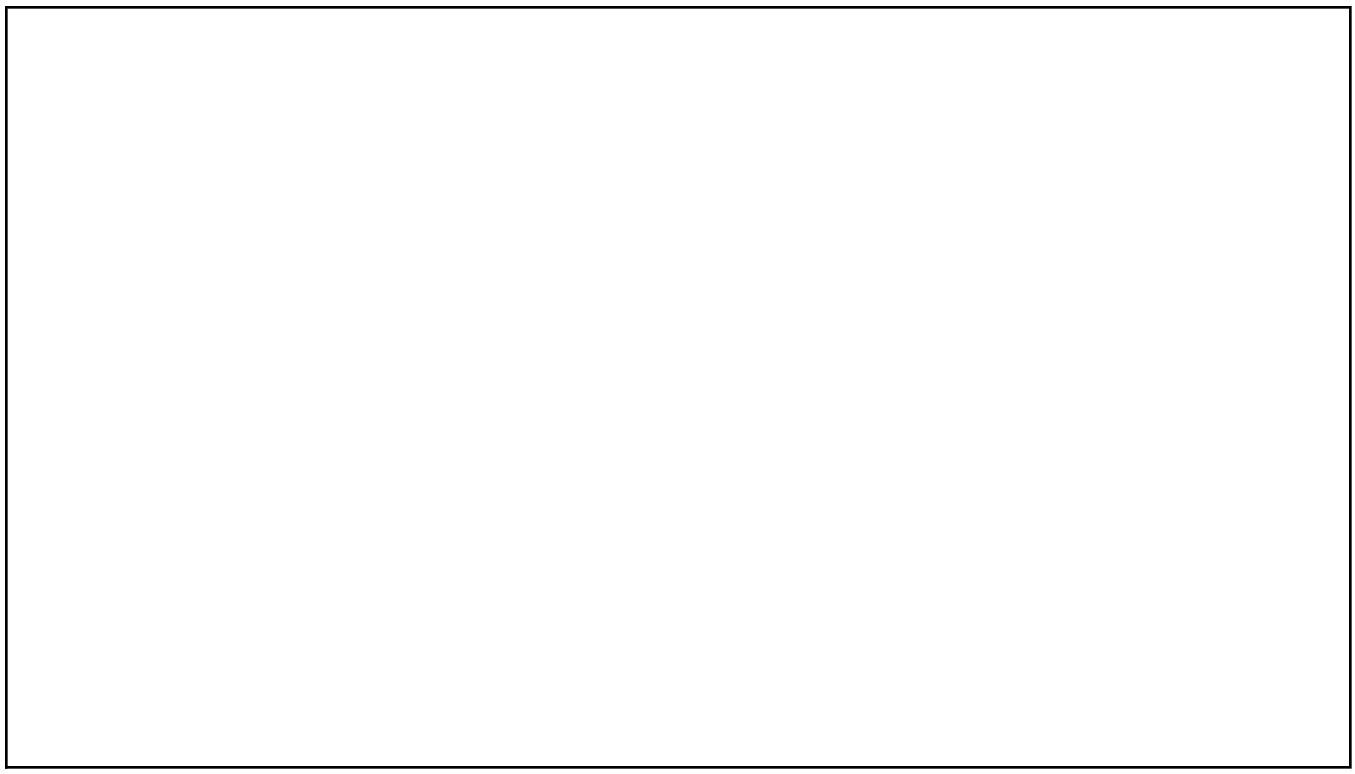
4. Propose a redesign in the form of a UML class diagram that applies your identified pattern to improve the design. **[5 Marks]**



5. Write code for the refactored version of main that uses your selected pattern and highlight the benefit of its application with relevant code comments. **[10 Marks]**

```
class Main {

  public static void main(String[] args) {
    String[] packages = {"Lite", "Plus", "Lite", "Lite"};
    AbstractFactory lite = new LiteFactory();
    AbstractFactory plus = new PlusFactory();
    AbstractFactory cur = null;
    //user code no longer coupled to bundles
    //the addition of new packages require only the introduction of a new factory
    for (String p: packages){
        if ( p == "Lite"){
            cur = lite;
        }else if (p == "Plus"){
            cur = plus;
        }
       System.out.println(
          p
          +" package has "+cur.createData().getDownload()
          +" download and "+cur.createTV().getChannels()
       );
    }
  }
}
```



```
// user code
class Main {

  public static void main(String[] args) {
    String[] packages = {"Lite", "Plus", "Lite", "Lite"};

    AbstractFactory fact;
    // no if statements, no dependencies of Bundles or even package classes
    for (String p: packages){
       // manager get the lite or plus factory based on the value of p
      fact = FactoryManager.getFactory(p);

      System.out.println(
         p+" package has "+fact.createData().getDownload()+
         " download and "+fact.createTV().getChannels()
       );
    }
  }
}
```

```
// module code
class FactoryManager {

  public static AbstractFactory getFactory(String type){
    AbstractFactory fact = null;
    //but the if statement is moved here
    //still open for modification for new packages
    //but at least it is isolated to this class
    switch(type){
      case "Lite":
        fact = new LiteFactory();
      case "Plus":
        fact = new PlusFactory();
    }
    return (AbstractFactory) fact;
  }

}
```

Question Code
Solution Code